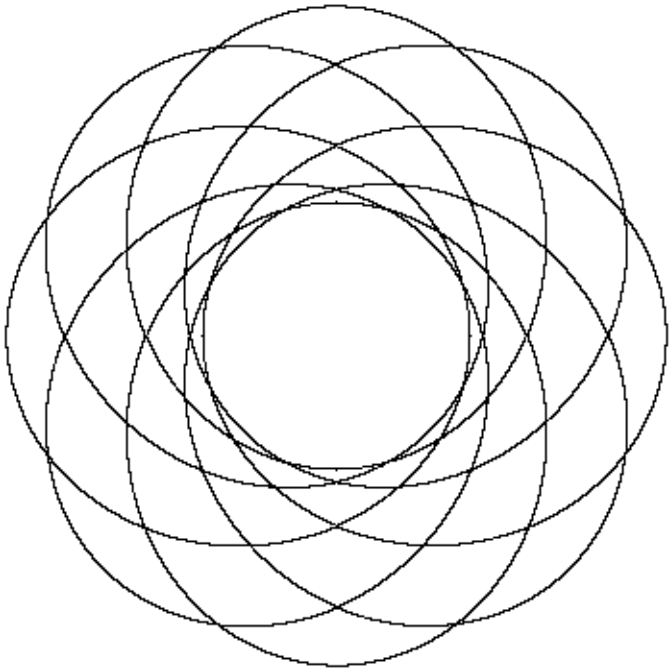
The XStar N-body Solver
Theory of Operation



By Wayne Schlitt

Table of Contents

Abstract

This document describes the N-body problem and ways that it can be solved. The "N-body problem" is the problem of trying to find how *n* objects will move under one of the physical forces, such as gravity. The N-body problem is developed mathematically and shown to be an Ordinary Differential Equation (ODE) and, using numerical analysis, solutions to the N-body ODE are constructed. Methods of evaluating the gravitational force function are also surveyed. Emphasis is given to details that are relevant to the program XStar, which graphically displays evolving N-body systems. The differences between the solutions to ODEs presented in numerical analysis texts and the requirements of the N-body problem in general, and the XStar program in specific are also discussed.

Introduction

The XStar program started out as a simple screen saver, but it evolved into a fairly large N-body problem solver. Likewise, this document started out as just the theory behind the XStar program, but it has evolved into a fairly complete overview of the N-body problem. While this document still emphasizes the aspects of the N-body problem that are relevant to XStar, it now covers enough of the N-body problem that the reader should have most of the background needed in order to understand the current research level papers on the N-body problem. Yet another way of viewing this document is as a case study in numerical analysis, as the N-body problem covers many of the important topics in this area.

This document concentrates on the “why” more than the “how”, and the options and trade-offs instead of the details and implementation. Formulas are only derived when it makes it clearer how areas are related; proofs are almost always skipped. These details can be obtained by using the referenced material. A background in integral and differential calculus and a college level physics course will be assumed, although someone without that background may well be able to follow most of the discussion. Knowledge of differential equations and basic numerical analysis would be a helpful, although not required.

Section 1.0 covers the issues involved with the N-body problem, from the mathematical foundations, to the important characteristics that a good solution should have. Sections 2.0, 3.0, and 4.0 each cover one of the three major areas where a N-body program can gain or lose efficiency. Sections 5.0 and 6.0 cover the conclusions that were found for the XStar program. For other programs, these conclusions may not apply, but they give an idea of what needs to be considered.

As far as the books and papers that are referenced by this document, the physics and calculus books can be replaced by any good college level text book without any loss in coverage. If you can’t get the required background of physics and math from a college text book picked out at random, then that text book isn’t any good.

None of the numerical analysis books seem to cover all of the details that need to be covered, and yet many of them contain in-depth proofs that are often beyond the scope of what needs to be covered. It appears that you will often need to reference a fairly large number of numerical analysis books in order to get adequate coverage (and explanations). The two numerical analysis books that stand out are *Numerical Recipes in C* (14), and the 1968 Schaum’s Outline Series’, *Theory and Problems of Numerical Analysis* (15). I find it somewhat depressing that the 60’s version of Cliff’s notes gives a better and more in-depth coverage of the problems of numerical analysis than most of the current numerical analysis books, such as *Burden and Faires* (2).

Andrzej Marciniak has produced a very good book *Numerical Solutions of the N-body Problem* (12). It covers mostly information in sections 1.0 and 2.0, but it gives very detailed (but very terse) proofs of all the items that this paper glosses over. Most other references to material covering the N-body problem are research papers that apply mostly to section 4.0.

Finally, a request from the author: the XStar program and this document were both created as learning exercises for myself, and in that light, I am very interested in hearing any feedback about either one. Everything from bug reports, notes about typos and grammatical errors, to any major misunderstandings or omissions are welcome. XStar contains several features that were donated by other people, and any new features to XStar or additions to this document will be seriously considered.

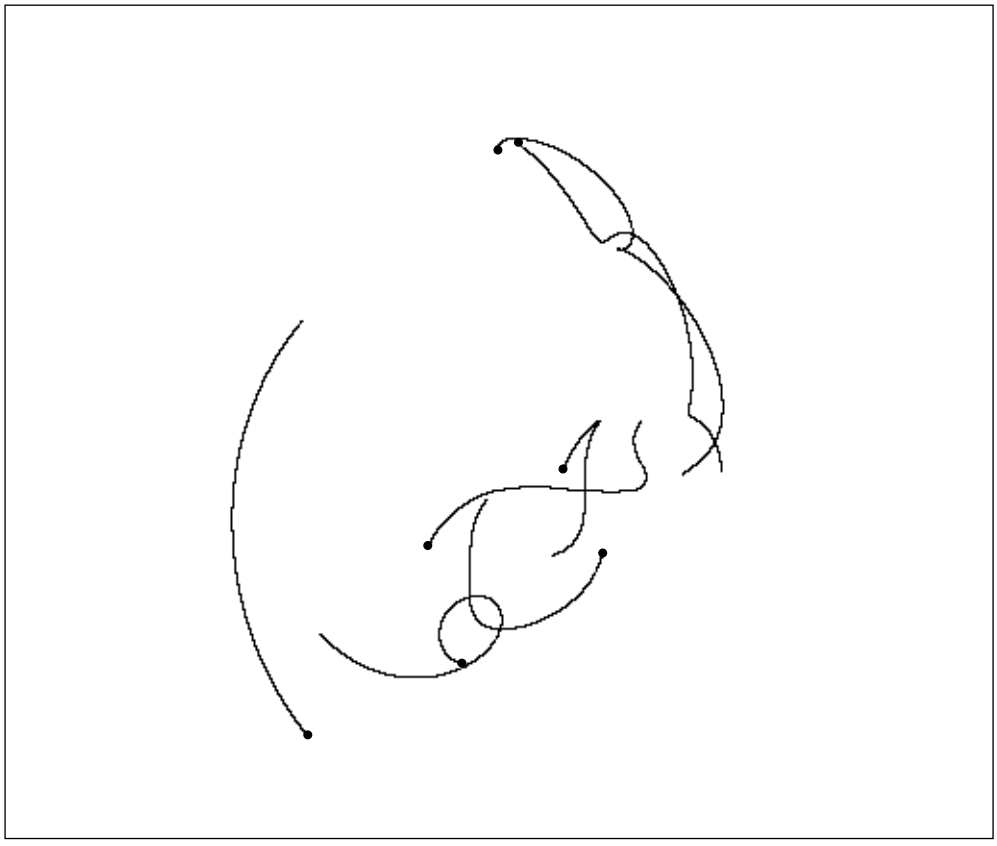


FIGURE 1. The Paths of seven stars, as created by the XStar program

1.0 Theory of the N-Body Problem

1.1 The Background and History of the N-body Problem

The N-body problem has been studied since ancient times, although people didn't realize that that was what they were doing. Any time there are several bodies (planets, stars, apples, electrons, etc.) that move under the force of one of the physical laws, you have an example of the N-body problem. When the ancient Greeks studied the movement of the planets, the changes of the seasons or how gravity works here on Earth, they were actually struggling to find solutions to the N-body problem. Significant headway on the problem did not occur until Copernicus and Kepler tackled the problem in the mid 1500's and it wasn't until Isaac Newton released his work, *Principia* in 1687 that a solution to the special case of $n = 2$ was found. In this book, Newton laid out his laws of gravity and the foundation of what we now know as classical physics. (3:148,12:1)

The study of the N-body problem has had a significant impact on history. The original theories of the movement of the stars lead to many religious and philosophical beliefs, some of which are still around today, for example, in the form of astrology. The work that Newton did on the N-body problem lead directly to the development of Integral and Differential Calculus. His development of physics has lead directly to modern engineering.(3:148) One of the major justifications for building the first electronic computer systems was to solve the N-body problem for artillery shell trajectories. (9:23)

Modern physics has found that there are only four fundamental physical forces, namely: gravity, electro-magnetic, strong nuclear, and weak nuclear[1]. These forces all have a few things in common: they can be expressed in very simple formulas, they all are proportional to some property of an object (mass, electrical charge, etc.), and they all get weaker the further apart the objects are from each other (7:30-33). How particles move under these forces are all solved with similar N-body programs. Gravity is the simplest of these forces, and the only one that is implemented by XStar and, so, it is the only one that will be directly discussed. Besides the modifications needed to handle the other forces, there are also specialized versions of the N-body problem that take Einstein's relativity into account, and ones that take into account quantum mechanics.

When most physics text books present these forces, they show the simple formulas, and they use these formulas to solve many simple cases. Rarely do they try and present cases where more than two objects are creating the forces and for good reason. As soon as there are three objects that move under one of the four fundamental forces, the N-body problem becomes very hard to solve. It is the N-body problem that links the four physical forces to the complex results that we see as the universe.(12:49)

1. Only gravity and the electro-magnetic forces are directly observed in everyday life. The strong force is responsible for the different elements that atoms come in and for nuclear power. The weak nuclear force causes certain types of nuclear decay.

Studying the structure of the N-body problem shows that when bodies move under one of the physical forces, the paths will be smooth and very predictable in the short run.(12:4) This is the source of almost all the predictability that we see in everyday life: objects fall down, rocks are hard, tides in the oceans, the seasons of the year, how machines function, etc.

Studying the structure of the N-body problem also shows that in the long run[1], it is very hard to predict how things will turn out and that very small differences in initial conditions can lead to wildly different results. This is the source of almost all of the chaos that we see in everyday life: the weather, the roll of a die, the way things crack and break, etc.

It is this conflict between the predictability and the chaos, the glimmer of almost knowing how a star pattern will turn out, but never being sure that lead me to develop the XStar program to its current form.

1.2 Newtonian Physics

Newton laid out the formulas needed to solve the N-body problem for gravity some 300 years ago. They are really fairly simple and the formulas are: (16:762-85,17:78-83)

x	The position of the body.
$v = x'$	Velocity is the rate of change of the position.
$a = v' = x''$	Acceleration is the rate of change of the velocity and is also the second derivative of the position.
$F = ma$	Force equals the mass times the acceleration.
$F = \frac{Gm_1m_2}{r_{12}^2}$	The force of gravity between two bodies (of mass m_1 and m_2) is equal to a constant G times the product of the masses, divided by the square of the distance r_{12} between the bodies. Technically, the formula looks more like $\vec{F} = \frac{Gm_1m_2}{\vec{r}_{12}^2}\frac{\vec{r}_{12}}{\lvert\vec{r}_{12}\rvert}$ where $\vec{r}_{12}$ is the vector between the two bodies and $\lvert\vec{r}_{12}\rvert$ is the length of the vector. That is, the force is projected along the line connecting the two bodies.

1. While it is *possible* to give precise definitions of “short run” and “long run”, it is not very useful to do so. Just be aware that what is the “long run” for some problems might be the “short run” for others.

Using the above equations, it can be proven that several properties of a star system can not be changed during the lifetime of the system. The proofs are fairly involved, but the results are well known. Collectively, these properties are known as the “constants of motion”.(12:49-56)

The constants of motion are:

- The total energy of the system must be conserved. So, if the kinetic energy of the system increases, the potential energy must decrease.
- Matter can be neither created nor destroyed.
- The total (linear) momentum of the system must be conserved.
- The total angular momentum of the system must be conserved.
- The center of mass of the system, if it moves at all, must move in a straight line and with a constant speed.

Knowing that these items must remain constant can be used to help determine if the results from a “solution” to the N-body problem is correct and, if not, the size of the error. It will be shown later that numerical solution can not, in general, be exactly correct, so determining the type and amount of errors is an important part of creating a good method for solving the N-body problem.

While these formulas are not very complicated, it can be hard to get a good feel for how the formulas respond with out working with them a fair amount, so looking at a few examples at this time is warranted.

1.2.1 An Example of Newtonian Physics With Just Two Bodies

As a first example, let’s look at the case of just two bodies in space as shown in FIGURE 3. Each body will have a position in space, a mass and a velocity, which are independent of all other bodies. If there is no force applied to a body, it will continue along on a straight line in the direction of the velocity vector. How quickly the body would move depends on the size of the velocity vector. In this example, Body 1 is moving up and to the left, Body 2 is moving down and to the left. Body 2 is also moving quicker than Body 1.

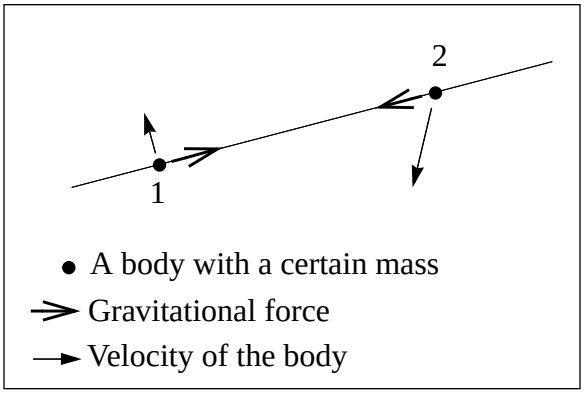


FIGURE 2. The components of a two body system

Newton's law of gravity says that there will be a force exerted by the two bodies on each other. The force will be along the line that connects them and the forces must be equal and opposite to each other. These forces will cause the bodies to accelerate toward each other and the magnitude of that acceleration depends only on the distance and mass of the other body. The acceleration will cause the velocities of the bodies to change and thus cause the bodies to no longer move in a straight line. (See FIGURE 3.)

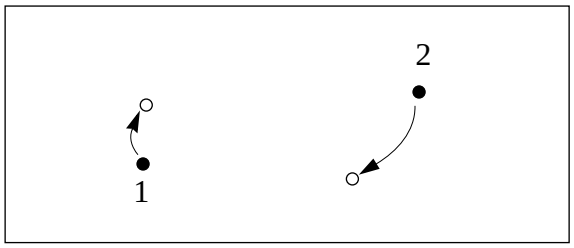


FIGURE 3. Paths caused by the gravitational force

When there are only two bodies in a system, they will always move on paths that follow one of the conic sections[1]. If the bodies are moving slowly enough that they orbit each other, they will move around their common center of mass.

1.2.2 An Example of Newtonian Physics With Four Bodies

For a slightly more complicated example, let's look at the case of four bodies as in FIGURE 4. Each body exerts a force on all other bodies. The total force, and therefore the total acceleration is simply the sum of all three forces. The paths that the bodies will take when there are more that two bodies can be very complicated. (See FIGURE 1.)

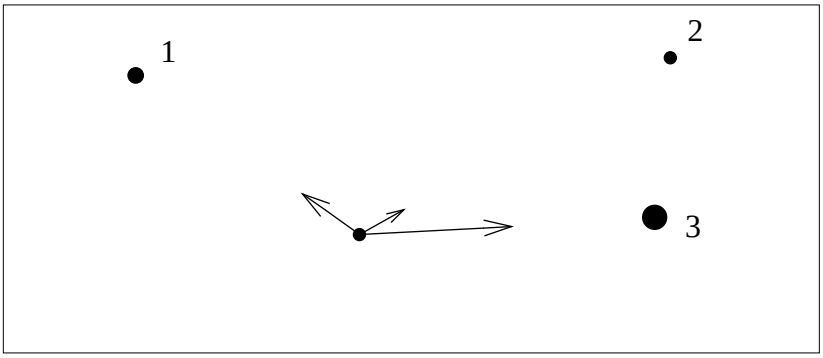


FIGURE 4. Forces exerted by several bodies

1.2.3 A Detailed Example of Newtonian Physics With Two Bodies

In the following example, we will look at a simple two body system in some detail One of the bodies will be so much more massive than the other body that it is effectively immobile. This is similar to the case of a comet and the sun, or the earth and a satellite.

1. The conics sections are the shapes made when you cut a cone with a plane. Depending on the angle of the plane, you can get a point, a single line, two intersecting lines, a circle, an ellipse, a parabola or a hyperbola.

FIGURE 5. shows the x and y positions of the comet with the sun located at (0,0). As the comet moves around the sun, the force of gravity is always pulling the comet toward the sun. The momentum of the comet keeps it moving in a direction that is slightly away from the sun. When the comet is closest to the sun, it will be moving the quickest and will have the most force exerted on it.

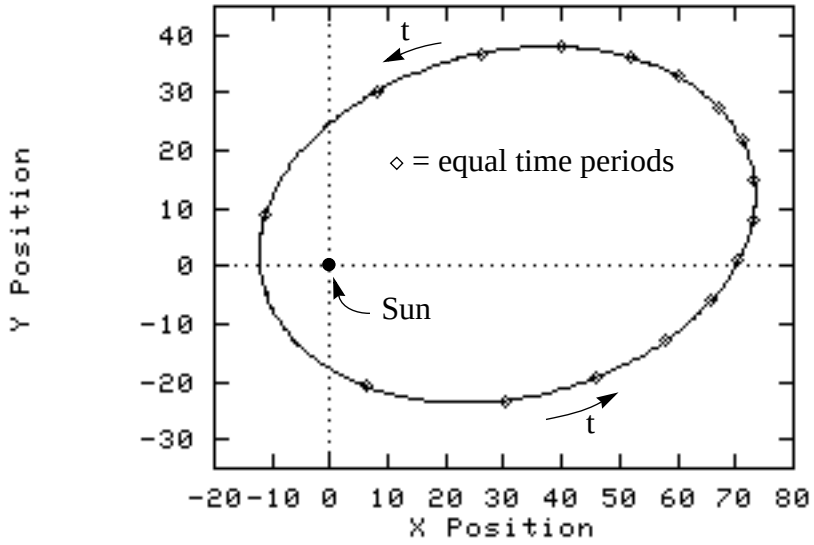


FIGURE 5. An X-Y plot of a two body system

While showing both the x and the y locations of the comet in one plot lets the orbit of the comet be seen clearly, it is also possible to show just the x or y component of the position as a function of time, as in FIGURE 6. and FIGURE 7. These plots let us see how quickly each component of the position is changing, as both the x and y component do not change at the same rate. As you can see, both graphs are very vaguely sinusoidal. Had the comet's orbit been a perfect circle, these graphs would have been perfect cosine and sine functions. The comet never gets within 10 units of the sun and yet the position functions already show some large changes in a short amount of time. Most comets are even more elliptical, which would cause the graphs to become even more distorted.

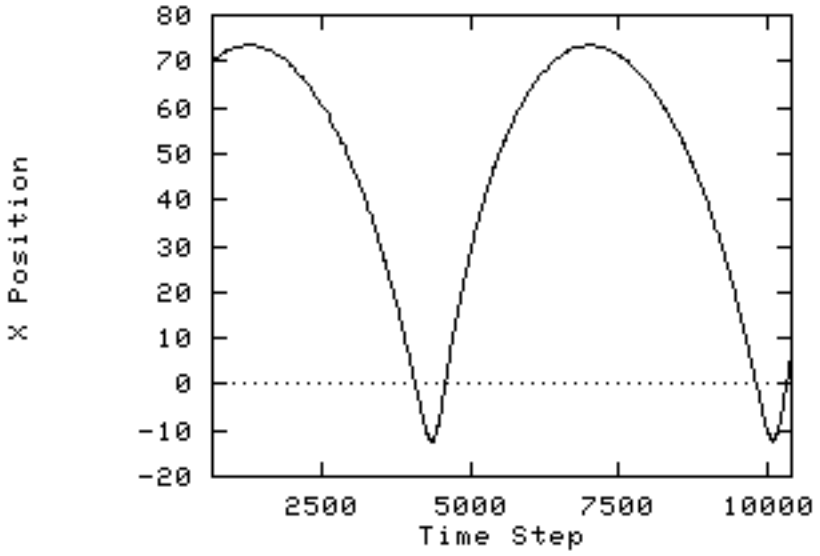


FIGURE 6. The X component of the position as a function of time

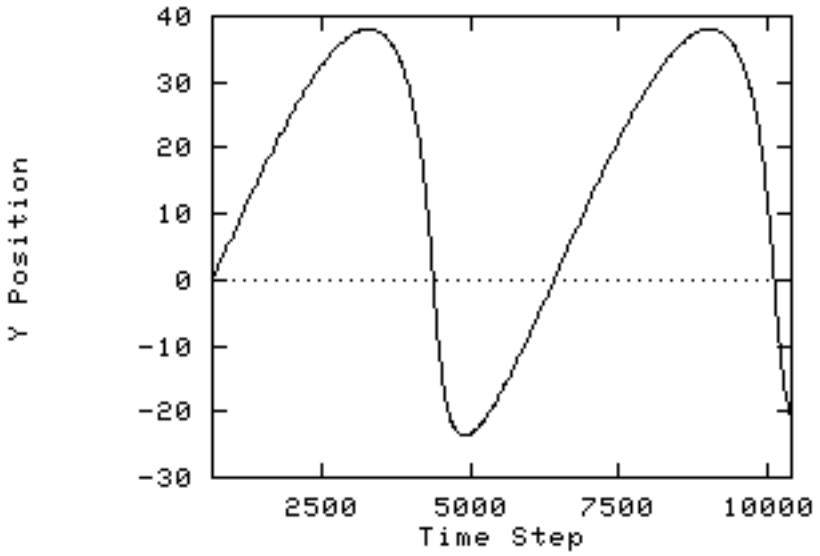


FIGURE 7. The Y component of the position as a function of time

As mentioned in the start of this section, the velocity of the comet at any given point of time will be the slope of the position curve. (That is, the velocity is the derivative of the position). The places where the position function is not changing (i.e. the tangent to the curve is a flat line) are the places that velocity function will cross the *t* axis. The places where the position function is changing the quickest is where the velocity function is the largest. Looking at FIGURE 8. and FIGURE 9. we can see that this is the case. Note that the velocity curves are even sharper than the position curves. Had the comet's orbit been a circle, the velocity functions would have been the sine and cosine functions.

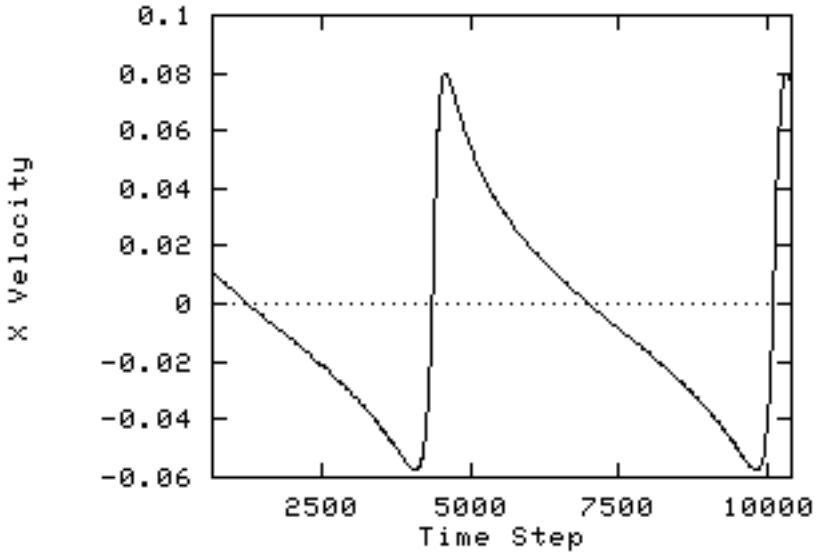


FIGURE 8. The X component of the velocity as a function of time

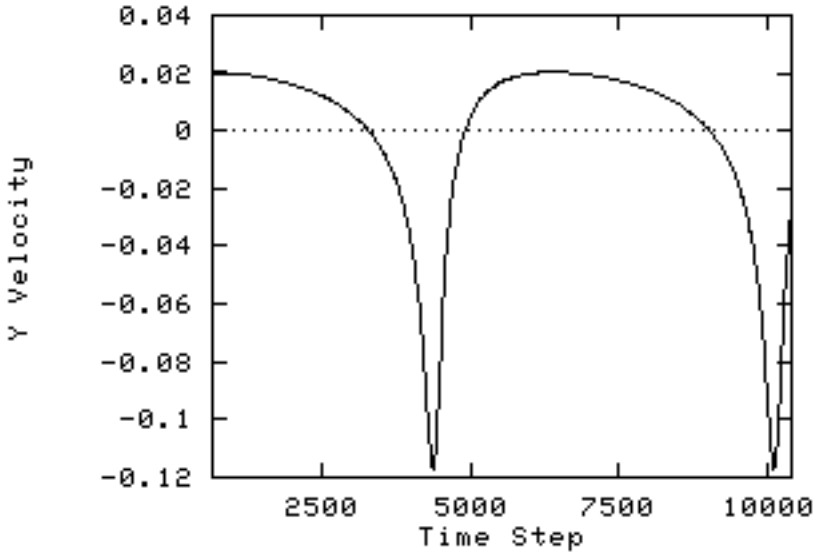


FIGURE 9. The Y component of the velocity as a function of time

Just like the velocity is derivative of the position, the acceleration is the derivative of the velocity. In FIGURE 10. and FIGURE 11., the components of the acceleration are shown. Just as was the case with the velocity, these graphs have sharper spikes and even longer flat spots. Again, the acceleration functions would have been the cosine and sine functions had the orbit been a circle.

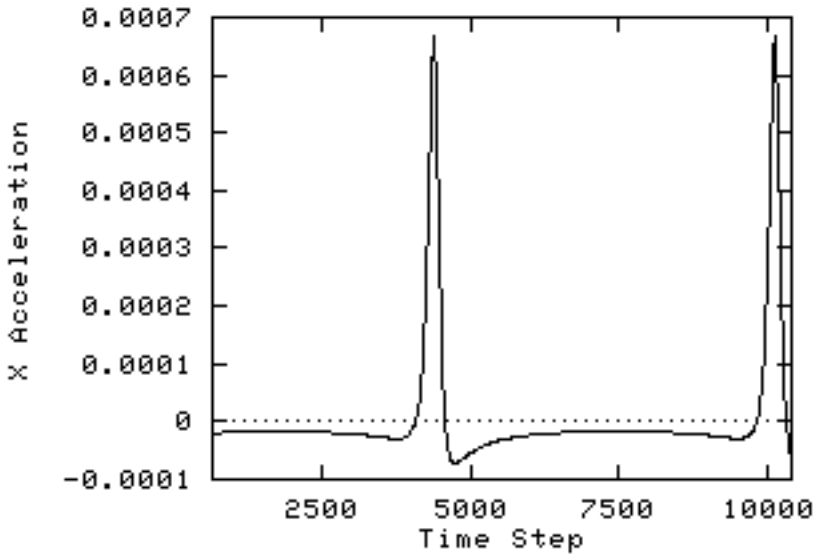


FIGURE 10. The X component of the acceleration as a function of time

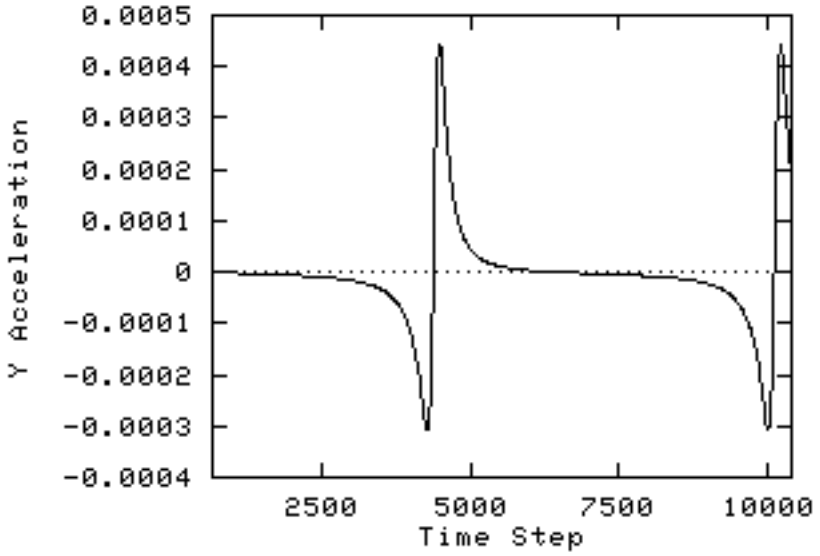


FIGURE 11. The Y component of the acceleration as a function of time

If, instead of just two bodies, we had plotted the case of four bodies, all of these functions would have become much more complex. A few things would have remained constant though: there would have been large time periods where the functions would have been very smooth and slowly changing, and there would have been short periods of times where large fluctuations would occur. While these sharp peaks represent a small amount of time, they represent a fairly large distance in the x-y plane. Take the time period from around 3900 to 4800 as an example. While it takes around 6000 time units for the comet to orbit the sun, this small 900 unit time period accounts for a surprisingly large proportion of the ellipse. (See FIGURE 12.)

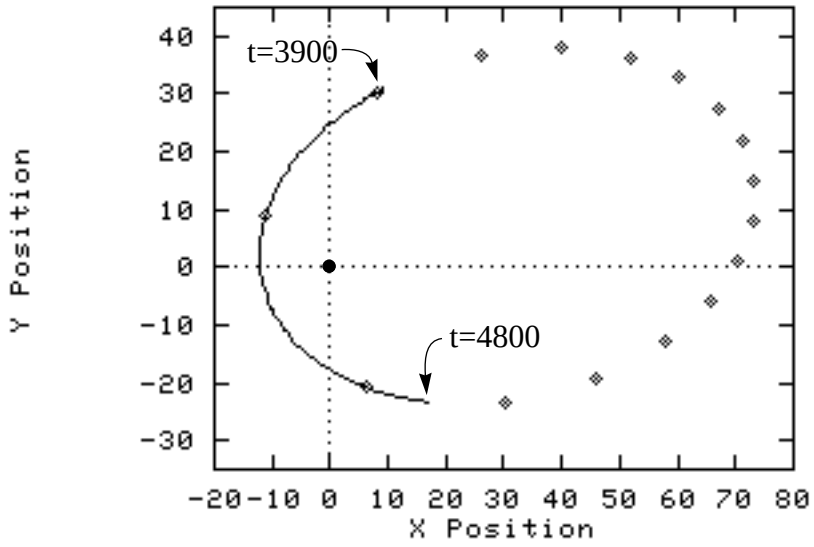


FIGURE 12. Path of the comet during the spikes

While the spikes in the graphs look very sharp, under closer inspection (See FIGURE 13.) we see that the functions are still very smooth. Later on, it will be shown these functions will always be smooth, when looked at close enough. There can never be any jumps or sharp bends in the functions.

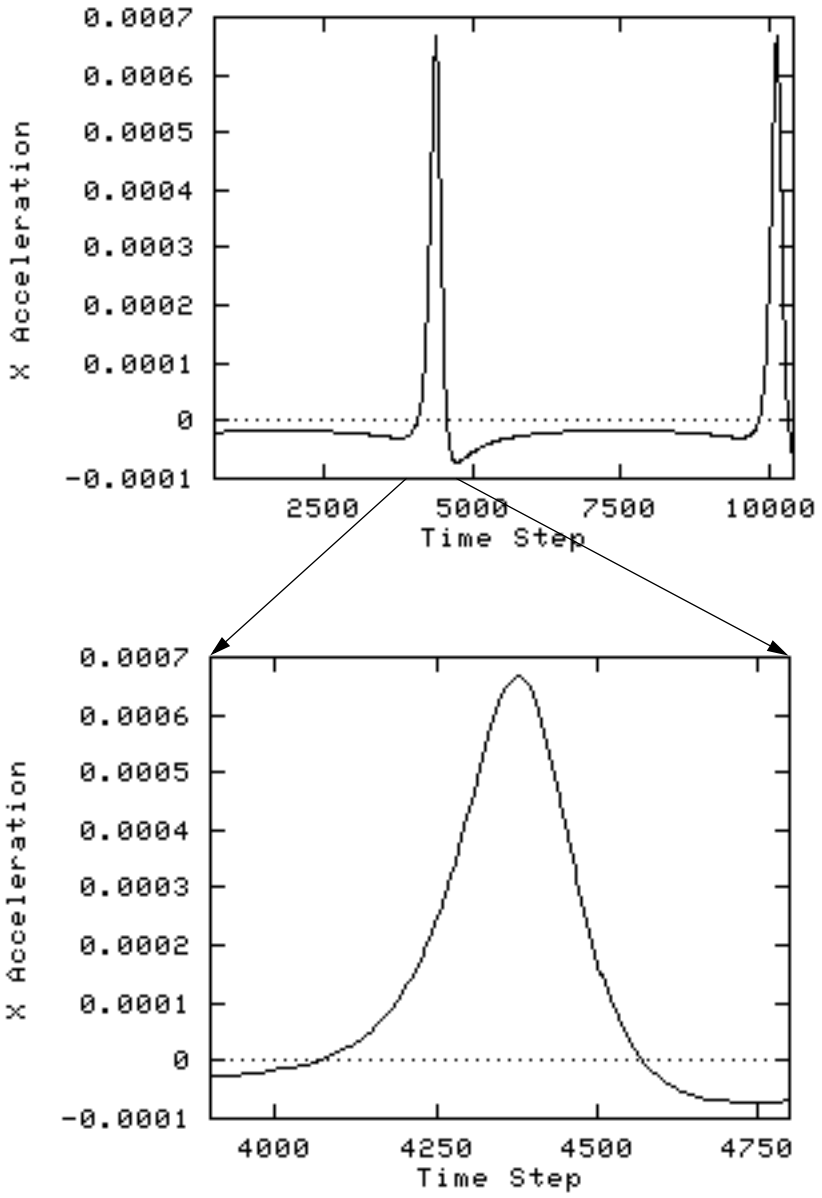


FIGURE 13. Functions are smooth, when looked at close enough

1.3 Development of Methods to Solve the N-body Problem

In order to calculate how the stars will move, we need to find the current acceleration based on where a star is at. A little bit of simple algebra on Newton's formulas shows that the acceleration for a given star due to one other star is $\vec{x}'' = \frac{Gm_2}{\vec{r}_{12}^{\,2/3}}\vec{r}_{12}$. For the N-body problem, one copy of the right hand term is needed for each other star in the system. The result is the vector equation: (12:50)

$$\vec{x}'' = \sum_{i=1}^{n} \frac{Gm_i}{\vec{r}_i^{\,2/3}}\vec{r}_i$$ Where $\vec{r}_i$ is the vector to the ith star.

While this is a vector equation, each dimension is handled in an identical manner and the dimensions are related to each other only by the change to the overall distance between the bodies. As was seen in Section 1.2.3, both dimensions could be looked at separately and both had similar functions. So, for the sake of simplicity, the vector nature of the N-body problem will be ignored for the rest of the document.

It should be noted that just because the vector nature of the problem can, for the most part, be ignored, the number of dimensions does have a significant impact on the nature of the N-body problem. When there is only one dimension, stars must move along a single line and therefore end up either colliding or going off to infinity. With two dimensions, stable orbits can be created, and it is possible for unstable systems to last for an indefinite amount of time. With three dimensions, the number of collisions is reduced even further. After all, in order for stars to collide, they have to be close in all three dimensions instead of just two.

The above formula is a messy enough as is, but as shown, it doesn't even taken into account the fact that the position, velocity and acceleration all vary with time. So, this equation will be abbreviated as:

$$x''(t) = f(x(t))$$ Where *f()* is the complicated force function

This equation is known as a "differential equation" because it relates the second derivative of x'', to the position x at time t. More specifically, this is known as an Ordinary Differential Equation, or ODE[1]. Sometimes there are ways of solving ODEs that come up with exact formulas, but for the N-body problem with more than three bodies[2], no one has found a method yet. When a problem can't be solved through the standard methods of differential equations, you generally have to resort to numerical analysis[3]. (12:49,2:233,1:289)

---

1. There are also Partial Differential Equations (PDEs), but they have no bearing on the N-body problem.

2. In 1912, K. F. Sundman developed a converging infinite series that solves the 3-body problem, however it converges too slowly to be of much use. The numerical methods turn out to be more efficient. (12:1)

3. Numerical analysis is the study of finding approximate solutions to equations through calculations as opposed to symbol manipulation like you would do with algebra.

The N-body problem is also known as an “Initial Value Problem” because, typically, the locations and velocities of the objects are known at a given time and the problem is to determine the state of the star system for some time in the future. It can be shown that the N-body problem satisfies the Lipschitz condition, and therefore given any particular initial situation, there can only be one possible outcome. While this might be expected for the N-body problem, it is not always true for all types of ODEs. (12:4)

One way of working with ODEs is to integrate out the derivatives, as follows. In the case of the N-body problem, the integration has to be done twice, once to get the velocity, once more to get the position. (16:1022,1:293)

$$\begin{aligned} x''(t) &= f(x(t)) \\ \Rightarrow \quad \frac{d^2}{dt^2}x(t) &= f(x(t)) \\ \Rightarrow \quad d^2x(t) &= f(x(t))dt^2 \\ \Rightarrow \quad x(t) &= \iint f(x(t))dtdt \end{aligned} \qquad \text{(EQ 1)}$$

The integration of $f(x(t))$ can then be performed by first breaking time down into a series of small, discrete steps, a process that is known as “discretization” of the problem. Then approximate solutions for each step in the series can be calculated and added together. This, of course, leads to small “discretization errors” which accumulate and can lead the future state of the bodies away from the one true future state. (14:567)

The goal is to make these errors be as small as possible and one way of reducing the discretization error is to reduce the size of each step. There are two problems with this solution though. First, it takes longer to come up with the final solution. Secondly, there is another source of errors, known as round off errors, which are inherent in (almost) all floating point calculations on computers. The smaller the step size, the more calculations will have to be done and the larger the accumulated rounding errors will be. So, there is a limit to how small the step size can be made without starting to increase the size of the total; error again. (12:25-6,1:308-9)

A second problem with integrating equation (EQ 1) is that it is hard to sample the value of $f(x(t))$ during the time period that we want to integrate over because the circular dependency between $x(t)$ and $f(x(t))$. This makes the standard methods of numeric integrate difficult to use. There are two basic approaches that can be used to work around this problem. The first method is to use an extrapolating or open ended method of integration where only the values of $f(x(t))$ from the past and present are used to integrate each step. The second method is to consider the integral in equation (EQ 1) as an unknown function. Even though we don’t know what this function is, at least the first few derivatives are easy to find, and it is theoretically possible to find all derivatives. This property means that the Taylor series can be used to approximate the function.

The simplest method of extrapolating integration is called Euler's method, and when applied to star movement with time broken down into steps of size h, the result comes out as:

$$v(t+h) = v(t) + h \cdot a(t) \qquad \text{(EQ 2)}$$
$$x(t+h) = x(t) + h \cdot v(t)$$

That is, at time $t+h$, the velocity will be equal to the velocity at time t, plus how much the star has accelerated over the time period, assuming that the acceleration is constant. Similarly, the new position is based on the original position and the velocity.

Equation (EQ 2) can be derived from (EQ 1) as follows:

$$x(t) = \iint f(x(t))dtdt$$
$$\Rightarrow \quad v(t) = \int f(x(t))dt$$
$$\Rightarrow \quad v(t+h) = \int_{-\infty}^{t} f(x(\tau))d\tau + \int_{t}^{t+h} f(x(\tau))d\tau$$
$$\Rightarrow \quad v(t+h) = v(t) + \int_{t}^{t+h} f(x(\tau))d\tau$$
$$\Rightarrow \quad v(t+h) \approx v(t) + h \cdot f(x(t))$$

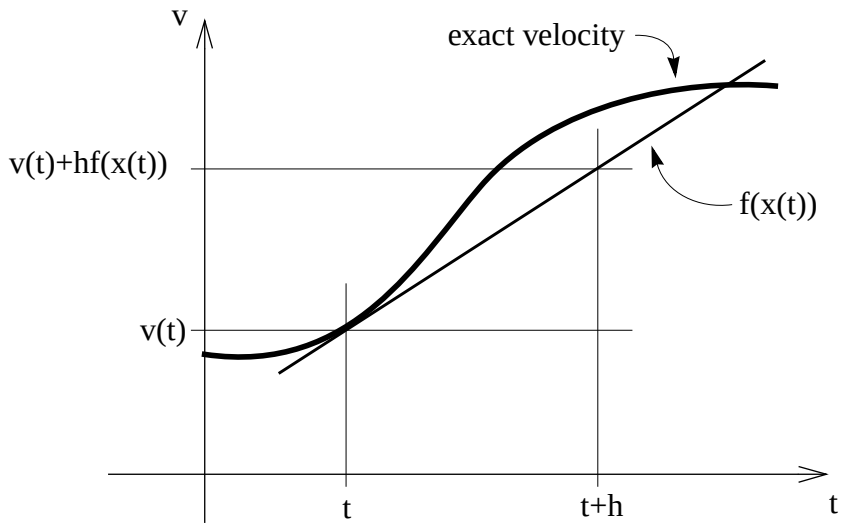


FIGURE 14. Graph of Euler's method

As FIGURE 14. shows, Euler's method can result in large errors because the acceleration *isn't* constant, it changes over the time period. Taylor's theorem says that the exact solution will really have the form:

$$v(t+h) = v(t) + ha(t) + \frac{1}{2}h^2 a'(\zeta)$$

So, there is really some other, unknown, term that is on the order as the square of the step size *h* times the value of the rate of change of the acceleration at some time ζ in the time period in which this move took place. Taylor's theorem says that $\zeta \in (t \ldots t + h)$, but that the exact value of ζ can not be known.

Using Taylor's theorem to derive a more accurate approximation of the function can be done by using additional terms. As an example, we could use:

$$x(t + h) = x(t) + hx'(t) + \frac{1}{2}h^2x''(t) + \frac{1}{2 \cdot 3}h^3x'''(t) + \frac{1}{4!}h^4x^{(4)}(t)$$

The error in this formula, i.e. the difference between the infinite Taylor series and the Taylor series that has been truncated[1] after 5 terms will be no larger than $\frac{1}{5!}h^5x^{(5)}(\zeta)$. (16:580-2) While the Taylor series could be used to create a very accurate solution, the problem with it is that even the third derivative of *x(t)* can be complicated and expensive to calculate in the case of the N-body problem.

All the other methods used to solve the N-body differential equation end up having a similar sort of error term. That is, the error will have some constant times, some power of the step size times, some complicated function evaluated during the time interval *(t ... t+h)*. The constant is often hard to calculate, the function is very hard to calculate, the time value that the function is evaluated at is impossible to calculate, so the only easily known quantity of this error term is the power of the step size. (16:580-2) The error term from above would normally be denoted as $O(h^5)$, that is, it is on the order of h^5. Thus, the formula is accurate to $O(h^4)$ and as a result, this method would be known as a "fourth order" approximation of the exact solution.

1.4 What Is the "Best" Method?

Before an evaluation of the different methods of integrating ODEs can be made, objective criteria must be defined. For most N-body problems, we can use this definition.

Definition: One method is **better** than another method if, in a fixed amount of time, it is able to calculate the positions and velocities with greater accuracy.

1.4.1 The Efficiency of a Method

In the definition of a "better" method, the phrase "in a fixed amount of time" is very important. It doesn't do any good to have a routine that can calculate the movements of the stars more accurately if it also takes longer to do the calculations than another method. The quicker method could be made more accurate by just decreasing the step size a little. For the same reason, it is also meaningless to just compare "which is more accurate at a given step size", some methods do much more work per step.

1. Discretization error is also known as truncation error for this reason.

There are three components of the efficiency that must be considered:

Firstly, how efficiently a method uses the results of the force function *f()* must be considered. As an example, Euler's is normally much less efficient than a fourth order Taylor series because in order to get the same accuracy with Euler's method, the step size would have to be much smaller and therefore, *f()* would have to be evaluated many more times. This turns out to be the predominate consideration for XStar.

Secondly, how efficiently a method selects the points in time to evaluate *f()* must be considered. The time step needs to be shortest when *f()* is changing rapidly in order to keep the discretization error to a minimum, but using the same step size when *f()* is changing slowly is inefficient. Dynamically changing the step size makes a method more complicated, and it also makes the results come out at an irregular rate. For some applications it is acceptable for a method to take considerably longer to return a result in certain situations than in other situations. XStar, however, needs to display the results in real time. The speed in which the star trails are displayed on the screen needs to reflect how fast the stars are moving.

Thirdly, how efficiently a routine calculates the function *f()* must be considered. A straight forward implementation of *f()* as outlined in Section 1.3 on page 17 produces a routine that takes $O(n^2)$ operations. So, when the number of stars is doubled, the routine will take four times as long to complete. There are methods of calculating *f()* that are $O(n \log n)$ or even $O(n)$, but they are much more complicated to implement. While this is very important when you are calculating the movements of the millions of stars in a galaxy, for XStar, this is not as important of a consideration.

1.4.2 The Accuracy of a Method

Another aspect of the definition of "a better method" is how do we determine which method "has the greatest accuracy?" We know that all methods that use numerical analysis to find a solution must have a degree of error in them, and that these errors accumulate. So, let's try this for an initial definition of accuracy:

Definition: One method is more **accurate** than another if the results more closely match what would happen in the real world.

This really is what the definition of accuracy should be, but there is a problem: how can the computers' results be compared to the real world? This would require experiments that would be very hard to perform. In Marciniak's book, *Numerical Solutions of the N-body Problem* (12:54-6), he uses a two body system that he can derive exact values for and compares the computers' results to these exact answers. However, it is very questionable that this method is suitable for judging the accuracy of 15 stars that are interacting, let alone millions of stars.

Even though the Lipschitz conditions says that there can be only one true answer, the unavoidable errors due to discretization and rounding make it is hard to tell which of several outcomes is really the “correct” one, or even if any of them are correct. If the “correct” outcome can not be obtained, then a choice must be made between which types of deviations are better or worse than other.

As a result, I have had to based my judgements of which methods are “better” by looking at the types of errors that I see and comparing them with other methods that have a smaller step size. If several different methods all agree that a particular star system should end up a certain way when a very small step size is used, then this information can be used to judge the methods when they use a larger step size. So the definition of accuracy that I have had to use is:

Definition: One method is more **accurate** than another if the results seem to more closely match the results of several other methods when they used “substantially” smaller step sizes, or more cpu time or both. Signs that a method has the types of errors that I don’t like cause that method to be downgraded.

This is not the rigorous, technical definition that I would like to have used, but I know of no better definition.

1.5 Types of Deviations In Star Movement

There are several different types of deviations from the ideal star movement that show up from the discretization and the round off errors. Most of these errors come from having one or more of the constants of motion changing over time.

1.5.1 Gaining or Losing Energy.

When a star gains or loses energy, instead of the star making an elliptical orbit around a collapsar, the orbits decay and either spiral in or spiral out. So, instead of a perfect ellipse, the star movement looks like FIGURE 15.

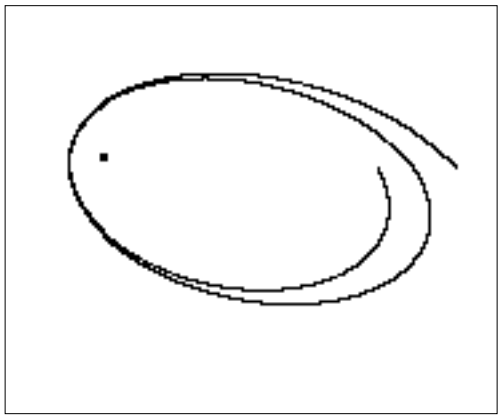


FIGURE 15. Results of a method that gains/loses energy.

This type of error tends to make a very uninteresting star system because a lot of stars will be lost either due to collisions or by stars moving off the screen. Euler's method tends to gain energy over time, but the fourth order Adam-Bashford method tends to very slowly lose energy.

1.5.2 Distortion

Some methods, such as the taylor3 method, both gains and loses energy, depending on the situation. When a star orbits a collapsar, it keeps it's elliptical orbit, but the orbit is more elongated than it should be. It is hard to even notice this unless you monitor the total energy level as the system progresses.

It is hard to predict what this type of error will do to a system. With only two stars it is hard to tell that this type of error even exists, but with many stars it is clear that the taylor3 method does not give as accurate of results as other methods.

1.5.3 Perihelion Shift

Some methods maintain the elliptical orbit around a collapsar, but the perihelion (the spot on the ellipse closest to the collapsar) rotates around the collapsar over time. Energy is neither gained nor lost over time, but the angular momentum changes.

The result of this error produces interesting results in XStar. Stars that orbit close to a collapsar form a solid disk of colors. (Should that make this error more acceptable than the rest?)

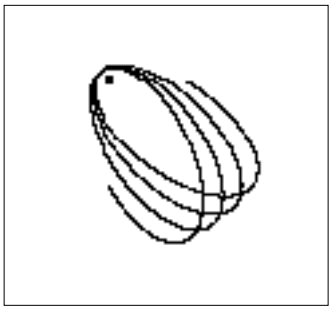


FIGURE 16. Results of a method that shifts the perihelion

1.5.4 Overstep Phenomenon

The overstep phenomenon is directly caused by the discrete movement of the stars instead of the continuous movement that exists in the real world. Take the case of a star that is falling toward a collapsar. As the star gets closer to the collapsar it picks up speed and with each step the force of gravity gets much stronger. When the star gets to time step t_4 (See FIGURE 17.) it will be so close to the collapsar that the next step will take it well past the collapsar onto the other side when, in reality, it should have collided with the collapsar. At time t_5, the star is now far enough away from the collapsar and moving at such a high speed that it will just keep on moving. The net result is that the star has gained a great deal of energy out of nowhere.

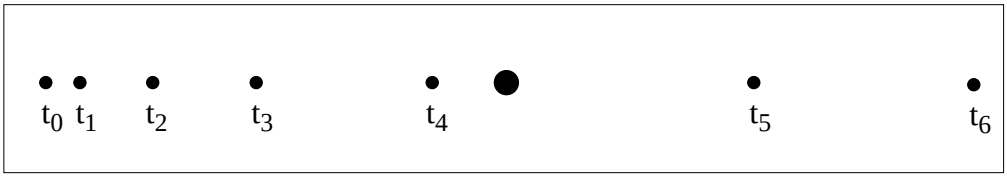


FIGURE 17. The overstep phenomenon

It is important to note that this same error can (and usually does) occur when a star is moving around a collapsar or star on a sharply curved path. Also, the star may not gain so much energy that it shoots off the screen. Instead, it can end up mixing this added energy into the rest of the stars, causing all stars to move toward the edges of the screen.

Mathematically this is the result of trying to integrate over the singularity when the distance to the collapsar is zero, or a pole in the complex plane when the star is on a curved path. The result is that the interval of convergence for the Taylor series is violated.

There are several ways of fixing this problem. The computer could try and check to see if a star moved through any other star after each step, but this would be very hard to do and it would be very expensive. The computer could simply say that the stars collide at a

fairly large distance, say around time step 3 in the above diagram, and this is what XStar currently does. Another option would be to use a different ODE integration method, such as the Runge-Kutta method, that handles singularities better thus letting the collision distance be smaller. Lastly, a “softening” factor can be applied to the system by adding a small constant to the distances that are calculated. For large distances this has little, if any, effect, but for cases where the particles might collide, this has the effect of turning the particles into clouds which can pass through each other.

1.5.5 Slingshot Effect

The sling shot effect isn’t an error, it is a real part of physics and it is has actually been used by the interplanetary space probes. However, the slingshot effect looks so much like the overstep phenomenon that it is very hard to tell the difference without actually watching the total energy levels of the star system.

This effect can happen only when there are at least three stars involved. When two stars interact, the kinetic energy that is gained over the period of time when the stars are moving closer together will always be exactly matched by the kinetic energy that is lost over the equivalent time period when the stars are moving away from each other. With three stars, it is possible for stars to convert some of the binding energy between two stars into the kinetic energy of a third star. This often happens when stars pass close to each other, with one star gaining a lot of energy. Since this is the same general situation that the overstep phenomenon occurs in, it is easy to confuse the two.

The case of using the sling shot effect with an interplanetary probe can make a good example. In FIGURE 18., a space probe uses Jupiter to gain kinetic energy after being launched from Earth. At time t_0, both the space probe and Jupiter are much more attracted (bound) to the Sun than they are to each other. The space probe, therefore, follows a roughly elliptical orbit away from the sun. At time t_1, the space probe has slowed enough that it is about to start falling back toward the Sun. However, Jupiter is close by and so the probe starts to “fall” toward Jupiter picking up a great deal of speed. Jupiter, on the other hand, is slowed down by a very tiny amount, and thus it will orbit the Sun in a slightly smaller circle. When the space probe passes around the front of Jupiter, it changes direction and starts to move away from both the Sun and Jupiter. Jupiter is still more influenced by the Sun than by the probe so it gets pulled away from the probe. Thus, the space probe loses less kinetic energy while it is moving away from Jupiter than it gained while it

was moving toward it. Without Jupiter being in the right place, the probe didn't even have enough energy for its orbit to reach Jupiters' distance from the Sun. After the slingshot effect, it now has enough energy to leave the solar system.

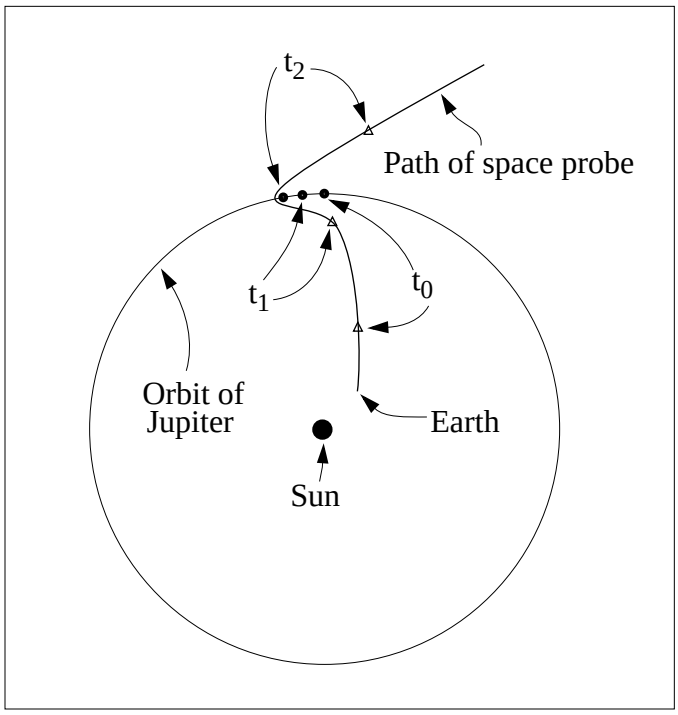


FIGURE 18. An example of the sling shot effect

1.5.6 Program Bugs

Errors in implementing programs are inevitable, but it might be hoped that with thorough testing, all serious programming errors can be eliminated. However, the nature of the N-body problem is such that it can be surprisingly hard to distinguish programming bugs from the unavoidable discretization and rounding errors. During the course of developing XStar, I found numerous bugs that did not have a significant impact *if* the step size was small enough. As a result, most bugs simply made a given ODE integration method appear to be less efficient than it should be. The number of bugs that were found only after making very careful comparisons between the results of the different methods lead me to believe that there probably still are some unknown bugs in at least some of the ODE integration methods implemented in XStar. Besides the possibility of unknown bugs, there are even a few known bugs that I haven't bothered to fix.

It is therefore important to realize that when two integration methods are compared, the comparison is really between two implementations of the methods with any number of bugs in either one. This comparison, therefore, can't be the final judge of which method is better, merely which implementation is better.

1.6 Speed-up Techniques

Section 1.4.1 on page 20 discusses some of the factors in evaluating the force function *f()*, but there are a couple of simple changes that can be done to even the straight forward $O(n^2)$ evaluation method. These changes can make a significant improvement in the speed of the program. First, because the forces between two stars are always equal and opposite, when the force is calculated for one star, the result can be applied to both. This cuts the cost in half, although it doesn't change it from being $O(n^2)$. Secondly, a time step value of 1 can be used which, in most methods, causes all the h^n's to drop out of the calculations. Thirdly, the gravitational constant G can be set to 1, saving a multiplication for every evaluation *f()*.

On the surface, these changes might seem to make XStar work in some unreal universe where gravity has a different strength but, under closer inspection, it can be shown that this is not true. The value of G is different in imperial units ($32\ ft/s^2$) and in metric ($9.8\ m/s^2$). Simply choosing the right units of measure and time can force G to be equal to one. XStar can still model the same physical systems this way, after converting everything from normal units to these "special" units.

As outlined in Section 1.3 on page 17, the accuracy of the ODE integration methods can increased or decreased by changing the step size h. With h now being defined as 1, how can the accuracy of the system be changed? The answer is that if you work through the units of G and its use in Newton's formulas, it can be shown that to make the star movements twice as accurate, you must cut the speed of all the stars in half and decrease the masses by a factor of 4. The smaller masses mean less acceleration, but the lower speeds keep the stars from flying off.

1.7 The Error Term

The final aspect of solving the N-body ODE that needs to be considered is the error term that was mention in Section 1.3 on page 17. Recall that there are three basic parts of the error term:

- some constant
- some power of h
- the evaluation of a derivative of *f()* at some point in the time interval from t to $t+h$.

That is, the error term has the form: $E = c \cdot h^n \cdot f^{(n)}(\zeta)$ where $\zeta \in (t \ldots t + h)$.

In Section 1.6 on page 27 it was said that the step size, h, should be equal to one. Since the constant c is often unknown, the derivative of the function is very hard to calculate and the time ζ is impossible to know, it appears that the only part of the error term that *is* known is h, which is now the constant 1. On the surface, this might seem to make the error term meaningless or impossible to use. Fortunately, this is only partially true.

Let's look at the taylor series expansion again. Recall that it is:

$$x(t+h) = x(t) + h f(x(t)) + \frac{1}{2}h^2 f'(x(t)) + \frac{1}{2\cdot 3}h^3 f''(x(t)) + \ldots + \frac{1}{n!}h^n f^{(n-1)}(x(t))$$

$$E = \frac{1}{(n+1)!}h^{n+1} f^{(n)}(x(\zeta))$$

$$\zeta \in (t \ldots t+h)$$

Now when $0 < h < 1$, each term of the taylor series gets smaller and smaller and eventually converge toward zero. When $h > 1$, the terms still converge toward zero because $\frac{1}{n!}$ will decrease faster than h^n will increase[1]. It might appear that more terms of the Taylor series would be required to get the same accuracy, however, for the N-body problem, this is not true. A star moving at 1/2 mile per minute (*h=1/2*) could also be considered to be moving at 30 miles per hour (*h=30*). Simply changing the units of measure doesn't change the physical system. The nature of the force function *f()* cancels out any changes in choice of *h* (as long as the velocities and masses are changed accordingly).

So does this mean that the order of the method is meaningless? That it is just as good to use a third order taylor series expansion as it is to use a 7th order Adam-Bashford method? Well, sometimes it is better, sometimes it isn't. The critical parts of the error term are the constant (which is unknown in some methods) and the value of the derivative of *f()* which is evaluated at the unknown time ζ. So, we can't say that one method is always going to be better than another method in all circumstances.

We can, however, look at how the error term changes as we change the step size. Say we have two methods, one of $O(h^2)$ and one at $O(h^3)$. If we double the step size, then the first method's error term will change to be $E_1(2h) = 8c_1h^3x'''(\zeta)$ while the other method will have $E_2(2h) = 16c_2h^4x^{(4)}(\zeta)$. When ratio $\frac{E_1(2h)/E_1(h)}{E_2(2h)/E_2(h)}$ is evaluated, we see that it is equal to 2. So, the higher order method *improves faster* than the lower order method. This also means that it will get *worse faster* as you decrease the accuracy.

The Taylor series' constant for the error term tends to be smaller than the other methods, and the Runge-Kutta's method tends to have smaller error term constants than the Adam-Bashford or Adam-Moulton methods. Thus for low accuracy levels, the taylor3 method is usually better than the ab7 method.

The last part of the error term, the evaluation of the n^{th} derivative of *f()*, is also important. Sometimes this value will be very small, sometimes it will be large. Sometimes it will cancel out the error from a previous step, sometimes it will make things worse. In

1. Consider the case where *n=2h*. Then h^{2h} will have *2h h*'s multiplied together, but *2h!* will also have *2h* numbers multiplied together and half of them will be larger than *h*. This is about the point where *n!* will be larger than h^n and thus $h^n/n!$ will be less than one.

Marciniak's book (12:72), his 7th order Adam-Bashford method gave much worse results than his 4th order Adam-Bashford method in a particular case. In my testing I found a case where the 4th order Adam-Bashford method with an accuracy parameter of one (i.e. the command 'xstar -m ab4 -a 1') gave much better results than the 7th order Adam-Bashford method with the higher accuracy parameter of -a 4. It also gave better results than -m ab4 with the higher accuracy parameter -a 2. Obviously, sometimes a method just gets lucky (or unlucky).

One more point should be made when determining which method of integrating an ODE might be better. Each time the derivative of *f()* is taken, a constant is pulled out due to the inverse square property of *f()*. So the first derivative of *f()* cause the constant -2 to be pulled out, the second derivative pulls out the constant -3. The constant of the error term of a 7th order method has to be multiplied by -9! = -362880. This is a very large constant that the higher order method has to over come and one of the reasons why lower order methods are more accurate when lower accuracy parameter settings are used.

2.0 Types of N-body ODE Integration Methods

There are many different methods that can be used to solve the differential equation $x''(t) = f(x(t))$ and thus can be used to calculate the movement of the stars. We have already mentioned some of them in passing, Euler's method and the Taylor series expansion in particular. There are also many others and in this section a few of them will be surveyed

In Section 1.4.1 on page 20, three areas of efficiency were discussed, the first of which was how efficiently a system uses the results of the force function *f()*. Section 1.7 on page 27 discussed how you can not simply look at the order of a method that solves an ODE to determine which method is best. Any one of these methods could be the most efficient method, depending on the circumstances of the particular star system.

A word of caution about the methods described in this section: the formulas as presented are usually playing fast and loose with notations and descriptions. In particular, they are usually presenting solutions to the equation $x'(t) = f(x(t))$ as this is how these formulas are normally presented in numerical analysis books, and also because it slightly simplifies the formulas. Converting these methods into something that can be used to solve the N-body ODE can sometimes be tricky.

As was mentioned in Section 1.5.6 on page 26, detecting program bugs can be very hard to do. As an example, I implemented the Runge-Kutta method (See 2.1.3) in what I thought was the correct method, but in Marciniak's book (12), he implemented it slightly differently. Instead of each step depending on the previous step, he had the second step depend only on the velocity. The third step was then dependant on the first step and the fourth and final step was dependant on the second step. Marciniak's implementation of the Runge-Kutta method turns out to be very slightly more efficient than mine. So, be careful if you try to implement an N-body program.

2.1 One Step Methods

The Euler's method, the Taylor series and Runge-Kutta's methods are all classified as "one step methods" because they calculate *x(t+h)* using only the initial starting point *x(t)* and consider only the time period from *t* to *t+h*. They solve the integration of *f()* over the entire evolution of the star system "one step" at a time. This is an important property and other methods, such as the multi-step methods, often use one-step methods to build up the "history" of *x* values. (12:14,1:302,366)

2.1.1 Euler's Method (-m euler1)

$$x(t + h) = x(t) + h f(x(t))$$

$$E = \frac{1}{2} h^2 f'(\zeta)$$

As we have seen before (FIGURE 14.), this is the simplest method, but the error term is only $O(h^2)$, making this method only $O(h)$. Thus, this method is only good for very quick and dirty approximations.

Sources: (14:569,15:197:19:413-6)

2.1.2 Taylor Series (not implemented)

$$x(t+h) = x(t) + h f(x(t)) + \frac{1}{2}h^2 f'(x(t)) + \frac{1}{2 \cdot 3}h^3 f''(x(t)) + \ldots + \frac{1}{n!}h^n f^{(n-1)}(x(t))$$

$$E = \frac{1}{(n+1)!}h^{n+1} f^{(n)}(x(\zeta))$$

We have seen this one before too and, while in theory it can be very accurate, it is not very practical because it can be very hard to calculate the higher derivatives of *f()*. It should be noted that Euler's formula is simply the Taylor series truncated after the second term.

Sources: (4:354-8,15:193,13:67-8,15:200-2,19:422-3)

2.1.3 Runge-Kutta's Method (-m rk4)

Runge and Kutta developed their method by trying to create formulas that can match the terms of the Taylor's series, but without the use of higher derivatives of *f()*. So, they looked at equations in the form of:

$$x(t+h) = x(t) + a_1 k_1 + a_2 k_2 + a_3 k_3 + \ldots + a_n k_n$$

Where

$$\begin{aligned} k_1 &= h f(x(t)) \\ k_2 &= h f(x(t + b_1 h) + b_1 k_1) \\ k_3 &= h f(x(t + b_2 h) + b_2 k_2) \\ &\ldots \\ k_{n+1} &= h f(x(t + b_n h) + b_n k_n) \end{aligned}$$

and the constants $a_1, a_2, \ldots a_n$ and $b_1, b_2, \ldots b_n$ need to be determined.

By using some auxiliary equations, an under determined systems of equations[1] can be found. Then, by choosing the values of a few of the constants a_i or b_j, the other constants can be solved for. Some values of the constants have better properties than others. In particular, by using different combinations of a_i constants and *k*-terms, it is possible to obtain two different Runge-Kutta formulas. As will be discussed in Section 3.0 on page 41, this has certain advantages.

The error term for Runge-Kutta's method can be hard, but not impossible to determine. It can be shown that the order of the error term is always going to be less than or equal to the number of *k* terms and if there are more than four *k* terms, the order must be strictly less than the number of *k* terms. So, using five *k* terms can not increase the order of the method, although it can reduce the constant in the error term. Thus, using four *k* terms is often the most efficient of the Runge-Kutta methods.

The most popular constants for the 4^{th} order Runge-Kutta method yield:

$$x(t+h) = x(t) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where:

$$k_1 = h\,f(x(t))$$

$$k_2 = hf\left(x\left(t+\frac{h}{2}\right)+\frac{1}{2}k_1\right)$$

$$k_3 = hf\left(x\left(t+\frac{h}{2}\right)+\frac{1}{2}k_2\right)$$

$$k_4 = h\,f(x(t+h)+k_3)$$

$$E = O(h^5)$$

The Runge-Kutta method uses a series of Euler like steps to create trial points into the interval that is being integrated. As is shown in FIGURE 22., the first evaluation of *f()* gets the slope of the path at time *t* and this slope is used to get a trial evaluation at the mid point of the interval. The second evaluation of *f()* is used to get a different evaluation at the mid point. The trial points are always taken at the location that is the result of starting at *x(t)* and moving along the given slope for the desired length of time. Even though k_2 and k_3 are both taken at the mid point, k_2 will have (in this example) a worse approximation of

1. A list of equations that has more variables (unknowns) than equations. When the system of equations is under determined you will always be free to pick (or be forced to pick depending on how you look at it) the values for some of the variables in order to find the value of the rest of the variables.

the correct slope because it is so far off the true path. Next, k_3 is used to get a trial evaluation at the end point of the interval. Finally, all four slopes are combined to get the final location.

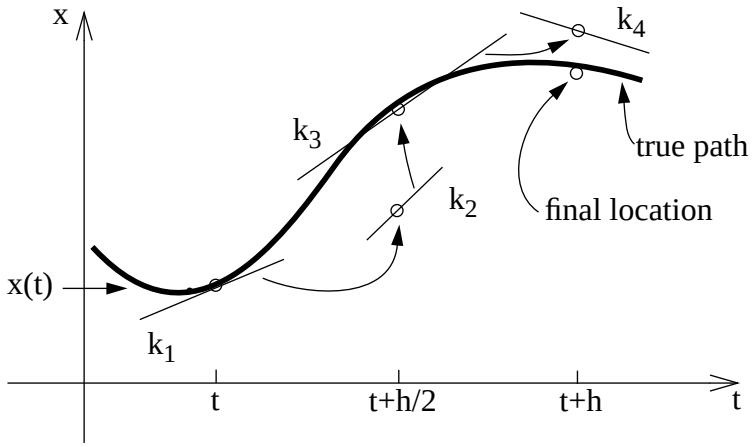


FIGURE 19. Graph of Runge-Kutta's method

Remember, it is not important for the Runge-Kutta method to have k_3 be more accurate than k_2, or for k_4 to end up the most accurate. Nor can you just take the trial points at random spots or increase the number of trials and have the results improve. The purpose of using these particular spots is so that when all four evaluations of *f()* are added together (with the right weightings), parts of the error terms of each step cancel each other out and the final result has an error term of a higher order than the individual Euler steps.

The problem with Runge-Kutta methods is that they require several evaluations of *f()*, which in our case is very expensive. To offset this expense the Runge-Kutta method must be at least four times as accurate. For XStar, this turns out to not be the case when compared to the multi-step methods.

Sources: (15:202-3,4:362-6,14:569-72,1:366-80,12:59-60)

2.2 Multi-step Methods

Multi-step methods use the previous values of *x* to help determine the future values. For XStar in particular, it seems a waste to just ignore the incredibly long history of star locations.

One of the disadvantages of the of the multi-step methods is that they require some sort of one-step method to get them started. Worse, the “getting started” phase is not limited to just the first time that the stars are created, but also when stars collide or bounce. So, bouncing star systems that use multi-step methods are often using the one-step “starting” method for a significant percentage of the time.

Another disadvantage is that the history of locations and velocities can take up a grate deal of space. Instead of just keeping the current and next values, multi-step methods often need to keep 2-4 times as much information. When working with a very large number of stars, this can make the difference between being able to fit everything in memory, and having to page parts of the storage to disk.

2.2.1 Modified Taylor Series Method (-m taylor3)

This method will not be found in any books, it is an ad hoc method that I created early in the development of XStar. Before I had even really looked into the books on solving this kind of differential equation, I knew that I could use the previous values to help out. An early version of XStar, version 1.1.0, used the formula:

$$x(t+h) = x(t) + hv(t) + \frac{1}{2}h^2 f(x(t)) + \frac{1}{6}h^3 f'(x(t))$$

But the $f'(x(t))$ term is expensive to calculate, so it was approximation by:

$$f'(t) = a'(t) \approx a(t) - a(t-h)$$

Simply adding this approximation made a dramatic improvement in the overstep problem when stars passed close together. The reason for this is fairly apparent: when an overstep occurs, the acceleration goes from being a large positive number to being a fairly large negative number. The difference is going to be a very large negative number. This dramatically reduces the amount of energy that is gained. The result looks like the star ‘just missed’ the collapsar.

One of my next thoughts was to try and approximate $f'(t)$ even closer by using additional terms. This is typically done by taking the derivative of the Lagrange polynomial. The Lagrange polynomial, *L(t)*, is the fairly straight forward polynomial that matches a given set of output values for a given set of values of *t*. Between those selected times, however, the values of the Lagrange polynomial can fluctuate widely. The more terms that the Lagrange polynomial has, the more it can fluctuate. Thus, when you take the derivative of the Lagrange polynomial, the result can get worse with additional terms.
(2:157-62,4:295-8,15:97,13:313)

I tried using the Lagrange polynomial of order 2 (i.e. the above formula), 3 and 4. The results were clearly the best for the 3rd order derivative.

2.2.2 Adam-Bashford's Method (-m ab7 and -m ab4)

The Adam-Bashford method uses any number of previous values to help calculate the next value of $x'(t) = f(x(t))$. This method has forms of any order, but they are all in the form of:

$$x(t+h) = x(t) + \frac{h}{r}[a_1 f(x(t)) - a_2 f(x(t-h)) + a_3 f(x(t-2h)) - \ldots]$$

Where r, a_1, a_2 ..., a_n, are constants.

At first glance, it might appear that these formula's would suffer the same problem that adding more terms to the derivative of the Lagrange polynomial suffered from, namely, the results would get worse with additional terms. After all, how much value is knowing the position from a long time ago really going to help? Some of the results in Marciniak's book seemed to confirm this suspicion (see page 28), but it turns out that the formulas do not get worse with additional terms. There does, however, get to be a point where the rounding error starts to get worse, so for practical reasons, the Adam-Bashford methods are normally limited to around the 7th order.

In fact, the 7th order Adam-Bashford formula is the default for XStar because it is the most efficient N-body ODE integration method for the default level of accuracy. The high quality results that XStar gets from the Adam-Bashford method is somewhat surprising considering how little coverage most numerical analysis books give it. Several don't cover it at all, others cover it only as part of predictor-corrector methods (discussed next). The results shown in Marciniak's book do not look very promising either. Still, for XStar, both the 4th order and the 7th order methods consistently ranked very high in terms of being the "best" method as defined in Section 1.4 on page 20.

Sources: (1:343-6,12:61,4:373-6,13:315,15:210-1)

2.3 Predictor-Corrector Methods

Predictor-Corrector methods use the idea that once you have an approximate value at a given time, there might be ways of improving this estimate. Predictor-Corrector methods seem to have a lot of folklore associated with them. It is possible to call the corrector method several times, but folklore has it that this is usually not worth while. Also, it is possible to have a predictor and a corrector with different orders, but the folklore says that the corrector should be equal to or only one order higher than the predictor. It is also said that predictor and corrector formulas usually have "the same form", although it is not clear exactly what is meant by that. (14:589-592,4:380)

According to most of the books, Predictor-Corrector formulas are now passe. Now a days people use either Runge-Kutta's or Gragg-Bulirsch-Stoer's methods. On the other hand, the books imply that the straight multi-step formulas were made obsolete by predictor-corrector methods, and I didn't find this to be true.

2.3.1 Modified Euler's Method (not implemented)

Euler's method assumes that the acceleration at time *t* is a good approximation for the acceleration for the entire interval from *t* to *t+h*. A better approximation would be the average of the acceleration at time *t* and at time *t+h*. This can be done by using Euler's method to predict the value of *x(t+h)* and then correcting it as follows:

$$y = x(t) + h\,f(x(t))$$

$$x(t+h) = x(t) + h\left(\frac{f(y) + f(x(t))}{2}\right)$$

$$E = O(h^3)$$

This method improves Euler's method from a first order to a second order formula at the cost of one additional evaluation of *f()*. While this is an improvement, it is still too low of an order to be useful.

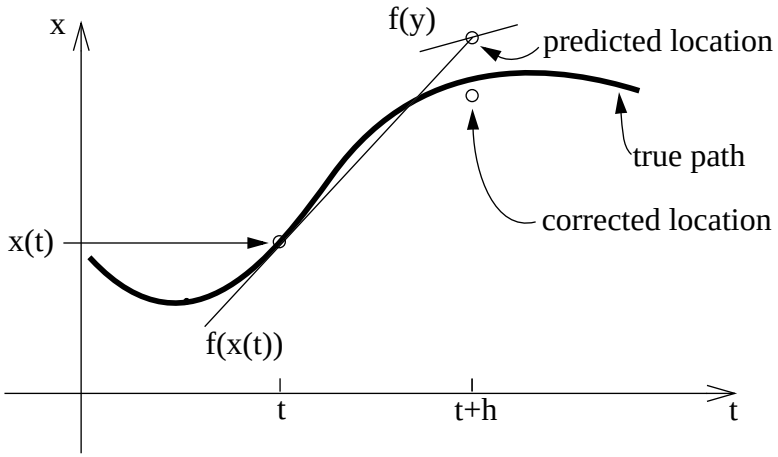


FIGURE 20. Graph of the Modified Euler's method

Sources: (13:70,15:207,19:417-8)

2.3.2 Adam-Moulton's Method (-m am7)

The Adam-Moulton's method is similar to the Adam-Bashford method except the formula uses the future value $f(x(t + h))$, instead of just previous values of *f()*. To get this initial estimate of $f(x(t + h))$, we use the Adam-Bashford formula as a predictor, and then use the Adam-Moulton formula to correct it. So the method has this form:

$$y = x(t) + \frac{h}{r}[a_1 f(x(t)) - a_2 f(x(t-h)) + a_3 f(x(t-2h)) - \ldots]$$

$$x(t+h) = x(t) + \frac{h}{s}[b_1 f(y) + b_2 f(x(t)) - b_3 f(x(t-h)) + \ldots]$$

Where $r, s, a_1, \ldots, a_n, b_1, \ldots, b_n$, are constants

Sources: (14:590,12:62-3,4:382-4,1:346-8)

2.4 Other formulas (Mid-point method) (none are implemented)

There turns out to be many other formulas of the form:

$$x(t+h) = a_1 x(t+n_1) + a_2 x(t+n_2) + \ldots + \frac{h}{q}[b_1 f(x(t+m_1)) + b_2 f(x(t+m_2)) + \ldots]$$

where the *n*'s and *m*'s may reference either into the future or into the past. Several of these have interesting properties.

For example, the formula:

$$x(t+h) = \frac{2}{3}x(t-h) + \frac{1}{3}x(t-2h) + \frac{h}{72}(191 f(x(t)) - 107 f(x(t-h)) + 109 f(x(t-2h)) - 25 f(x(t-3h)))$$

$$E = \frac{707}{2160} h^5 f^{(5)}(\zeta)$$

This formula is similar to the Adam-Bashford formulas, except that it uses the previous positions as well as the previous velocities and as a result it has a smaller error term.

One of the more useful formula is the mid-point formula:

$$x(t + h) = x(t - h) + 2h\, f(x(t))$$

$$E = \frac{1}{3}h^3 f'^{(3)}(\zeta)$$

This formula is no more expensive to calculate than Euler's method, but it is of $O(h^2)$ instead of $O(h)$. The mid-point method is named because it is saying that a good estimate for the average acceleration is at the middle of the time period.

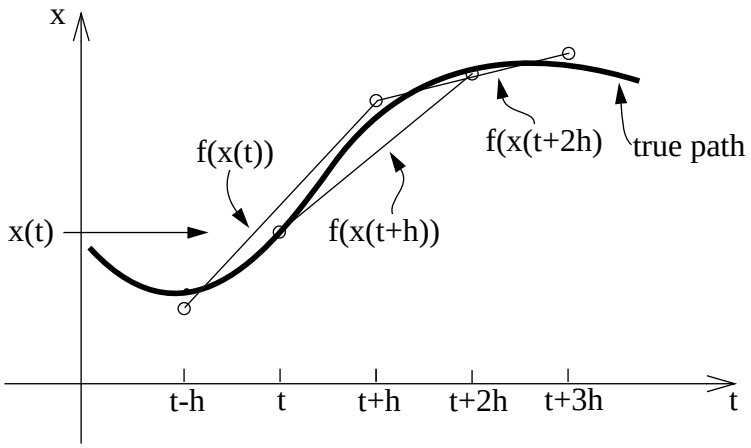


FIGURE 21. Graph of the Mid-Point method

The down side of the mid-point formula is that it is not very stable. The locations of the even time periods *(t, t+2h, t+4h)* depend only on the other even time periods and they are only loosely coupled with the odd time periods via the slopes[1]. (The same is true for the odd time periods.) So, it is easy for the time periods to get out of sync and for the even and odd time period points to start moving along totally separate paths. For this reason, it is not recommended to use the mid-point method for a large number of steps.

Sources: (1:320-5,4:375-6,14:580)

2.5 The Gragg-Bulirsch-Stoer Method (-m gpemce8)

The Gragg-Bulirsch-Stoer method is literally in a class by itself and seems to be the current favorite for solving this type of differential equation.

The basic idea behind this method is that as the step size is decreased, the answer should become more accurate. If a large interval *H* is taken and a variety of smaller step sizes are used over that interval, then you will get a variety of different answers but the answers should converge toward the correct outcome as the step size gets smaller. (See FIGURE 22.) Well, why not take these converging answers and create a polynomial

1. This method is also known as the "leapfrog" method because of the way even and odd time periods leap-frog over each other.

approximation of the answers as a function of the step size? Then the magic step size of *h=0* can be put into this polynomial, and the result should be a very good approximation of the answer as if no discretization had ever been done. (In FIGURE 23., the polynomial is shown as a function of the number of steps, but to make it a function of the step size is fairly straight forward.) This same basic idea is also used in Romberg's integration, and the whole idea was pioneered by Richardson with his idea of "extrapolating to the limit."

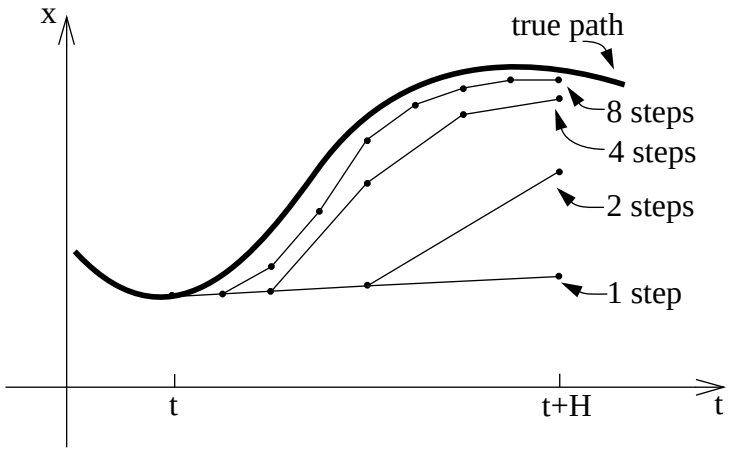


FIGURE 22. Results of using smaller step sizes

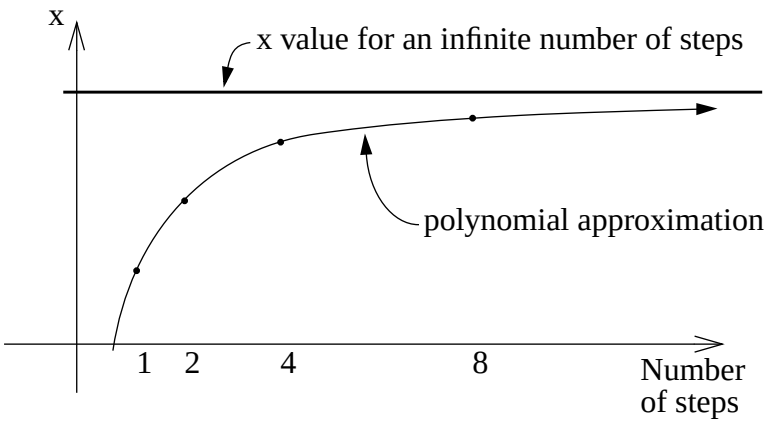


FIGURE 23. Polynomial approximation of x based on step size

The large step size H can be so large that the results from using even the largest number of steps (8 in our example) does not have to be very accurate. When all the trial movements are combined to form the polynomial approximation, the end result will be an extremely accurate method. So, even though each movement of step size H requires a large number of evaluations of the force function *f()*, it is still much more efficient that most other methods. It can also be made to be much more accurate than almost any other method because the final result does not suffer from the rounding errors that would be created by using incredibly tiny steps in other methods. For this reason, the Gragg-Bulirsch-Stoer method is clearly the best method when extremely accurate calculations must be done.

There several pieces to making the Gragg-Bulirsch-Stoer method work:

First, Richardson's idea that the final answer can be thought of as a function of the step size and this function can be used to extrapolate to what the answer would be if step size was zero.

Secondly, Gragg figured out that the mid-point method mentioned above, slightly reworked, has error terms that has only even powers of *h*. By careful manipulation when creating the polynomial approximation, it is possible to cancel out two orders of the error term at a time.

Thirdly, Bulirsch and Stoer figured out that using rational approximation instead of Gragg's polynomial approximation allowed you to break the limits of convergence of the taylor series around singularities and poles in the complex plain. They also figured out a good method of decreasing the step size (the Bulirsch sequence).

Lastly, Marciniak's method also implements a method of "Discrete Mechanics". That is, this particular implementation tries to keep the constants of motion constant at the expense of keeping the positions and velocities close to the exact answer. Since it is often impossible to tell what the exact answer should be, having the system at least behave well in other respects can be very useful.

Combining all these things together yields a very accurate, very fast and *very* complicated method. It is so complicated in fact, that this is the only method that I didn't write the code for myself. Instead I used the example code out of Marciniak's book, fixed a few bugs and made a few tweaks. The only problem is that Marciniak only implemented Gragg's method of polynomial extrapolation instead of using rational extrapolation. When XStar uses this method and two stars pass close together, the combined use of polynomial approximation and the discrete mechanics can give spectacularly wrong results. Try 'xstar -m gpemce8 -a .25' sometime...

Sources: (14:582-8,12:121-4,129-30)

3.0 Variable Step Size N-Body ODE Integration Methods

The second area of efficiency that was discussed in Section 1.4.1 on page 20 was that you don't have to select equally spaced points in time to evaluate the force function *f()*. As was seen in Section 1.2.3 on page 10, the force function *f()* has periods when it is changing very rapidly and periods when it is very smooth. The system needs the most accuracy when a star is making a sharp curve around another star, and it needs the least accuracy when a star is slowly moving off in a fairly straight line.

Many of the ODE integration methods, the Runge-Kutta method and the Gragg-Bulirsch- Stoer method in particular, can estimate the accuracy of the results that they are returning and it is possible to have them adjust the step size accordingly.

The key to creating a variable step size ODE integration method is to find a bound on the error term. If the error term is too large, then you need to decrease the step size. On the other hand, if the error term is too small, then you should be able to increase the step size without causing any serious problems. The CPU time you save can be used when a small step size is really needed.

The major problem with variable step size methods in XStar is that they tend to have the smallest step size, and thus run slower, when the stars are actually moving the fastest. If you implemented a variable step size method and didn't do anything to counteract this, you would see stars "slow down" as they approached collapsars and "speed up" when they are far away from them. For this reason, no variable step size methods were considered for XStar.

Another consideration when using variable step size methods is that the most well known implementations decrease the step size for all stars in the system anytime any of the of the stars need greater accuracy. When there is even a moderately large number of stars, say around 20-40, there will almost always be at least one star that needs a small step size. Implementing separate step sizes for each star can be complicated because you can't update the locations/velocities all at once. Some stars many need to be updated many times before some other star gets updated once, and until a star has been moved you can't calculate the force function.

One method of getting around the problem of all stars not being updated at the same time is to create a polynomial approximation of the path and any time you need to find the location of a star, you evaluate this polynomial. (10:6) In order to create this polynomial, a history of previous locations must be kept. This history can then be used in multi-step ODE integration methods. Normally the multi-step formulas are much harder to convert into a variable step size method due to their requirement of keeping history information, but when each star has it's own step size, this is no longer as much of a handycap.

Care must be taken not to try and make the lower bound of the error too small. The computer can always make the error value equal to zero by making the step size so small, that the stars won't move at all. Instead of using a global lower bound on the error, it is often better to make it relative to the size of velocity, or some other flexible formula.

3.1 Method of Changing the Step Size

As was shown in the discussion of the Gragg-Bulirsch-Store method in Section 2.5 on page 38, decreasing the step size generally increases the accuracy of the result. The basic idea is you can take just about any method discussed in section 2.0 and evaluate each interval twice, once with a step size of *h*, and once with a step size of *2h*. The difference between the answers can give us an estimate of the error term.

While it might seem awfully expensive to evaluate each interval twice, just to see if step size needs to be changed, there are several mitigating factors that make this method very practical. First, as has already been mentioned, the savings you get by making a very large step size when the stars are making smooth paths can pay for a lot of overhead. Secondly, the initial evaluation of *f()* can be used for both the movement with step size *2h* and for the first half of the two steps of size *h*. Lastly, you can usually take the results of the error estimate and use it as a correction factor and come up with an answer that is more accurate than either of the two trial movements.

3.2 Method of Changing Orders

Many methods, such as the Runge-Kutta and the Adam-Bashford methods, can have implementations of almost any order you could want. Another way of estimating the error is to take a movement step once with a method of $O(n^a)$ and once with the same method only using an order of $O(n^{a+1})$. Like the method of changing step sizes, the difference between these two steps can give you an estimate of the error for that step.

The multi-step formulas are particularly applicable to this method since all you have to do is keep a slightly longer history of the star movements, no additional evaluations of *f()* are required. The coefficients to the multi-step formulas are fairly easy to calculate and you can even create a table of them and vary the order of the formula as you change the step size.

The Runge-Kutta is harder to make efficient, but by carefully selecting the points that are probed, it is often possible to find ways of combining terms in two different ways to give two different orders. Fehlberg found six k-terms that could be evaluated two different ways, one gives a 4^{th} order method and the other gives a 5^{th} order method. Verner created a set of 8 k-terms that creates a 5^{th} and 6^{th} order pair.

3.3 Method of Changing the ODE Integration Method

By using two different ODE integration methods with different error terms, you can also get an estimate of the error. As an example, the Euler method and the Mid-point method can both be evaluated at the same point without having to evaluate the force function an additional time. Comparing the results will give you an estimate of the error term. A more useful example is with the predictor-corrector methods. The difference in the results between the predictor and the corrector can also give you an estimate of the error term.

3.4 Method of Internal Checking

Since the Runge-Kutta method takes several trial points into the future, it is possible to get a crude estimate of the error by looking at the differences between the evaluation of *f()* at these points. For the standard 4^{th} order Runge-Kutta method, one reference (11) states that $\left|\frac{k_2 - k_3}{k_1 - k_2}\right| \leq 0.02$ is a good estimate for when to change the step size.

For the Gragg-Bulirsch-Stoer method, it is common to keep increasing the number of steps until a good rational approximation can be obtained.

4.0 Efficient Force Function Evaluation Methods

When there are large number of stars, the evaluation of *f()* can be very time consuming. If the formula in Section 1.3 on page 17 is implemented in the obvious fashion, for each star, the distance to every star must be found, a process that takes on the order of n^2 operations. This particular method of evaluating the force function is known as a Particle-Particle method because each star is treated as a particle and it is checked against other stars (particles). If you have say, 100 stars, then there will be 100 evaluations of the force function f(), each of which will have 99 terms that involve the expensive operations of division and taking a square root. For Xstar with a system of 100 stars, the evaluation of the force function will account for well over 75% of the CPU time used. The integration taking up around 10% and all the other housekeeping and display code taking up the last 15% of the CPU. Even for the default configuration of only 15 stars, the evaluation of the force method accounts for almost half of all CPU time used. For other N-body programs that have to evaluate thousands or even millions of bodies, this method of evaluating the force function is almost completely useless.

Because the efficient evaluation of the force function *f()* is so critical for most N-body programs, this has been a very active area of research in the last 15 years. There are many different approaches, but so far there doesn't seem to be any particular method that has become the clear favorite among the N-body research community. While there is also a wide variety of ODE integration methods, they at least have well known trade-offs and areas of strengths. The trade-offs among the force evaluation methods do not appear to be as well known.

The list of methods of evaluating the force function contains many exotic names, such as: (8)

- Particle-Particle
- Particle-Mesh
- Particle-Particle/Particle-Mesh
- Tree-Code Top Down
- Tree-Code Bottom up
- Fast-Multipole-Method
- Tree-Code Particle-Mesh
- Self-Consistent Field
- Symplectic Method

Since this is still an area of active research, this document will not try to give a complete overview of this area. Instead, a few techniques will be discussed and interested readers are encouraged to read the internet document ftp://ftp.amara.com/papers/nbody.txt (8) which is a very good starting point for finding more information in this area.

4.1 Floating Point Rounding Errors and the Force Function

In Section 1.3 on page 17, there were two types of errors that were mentioned. First, there is the discretization error and the minimization of that error was the chief concern of Sections 2.0 and 3.0. The other type of error that was mentioned was floating point rounding errors, but it was noted that as long as the step size was not extreemly small, that this was not a primary concern. In the evaluation of the force function, this is not the case and the effects of rounding errors can be quite pronounced. For this reason, a brief discussion of the cause and effects of rounding errors is in order.

Internally, computers keep floating point (non-integral) numbers in a format that has a fixed number of significant digits, while the decimal point can float to the left or right. Most modern computers can have at least 15 significant digits, but for easy of illustration, let us assume that the computer can only store 4 significant digits. So, we can have numbers such as 3.141, 100, 0.0000456, and 1E+99, but we can't have numbers such as 3.1415, 100.01, or 0.000045678. When the computer adds two numbers together, often the result has more than 4 significant digits. In order to store result, it is necessary to round the result to 4 significant digits.

As an example, say the computer has to add 3.141 to 100. The exact result would be 103.141, but that has 6 significant digits. The computer would then round the result to be just 103.1, losing the 0.041 from the result. It may not be obvious, but this means that addition is no longer associative. Say the computer was to calculate 0.05 + 3.141 + 100. If the computer adds the first two numbers together, it will get 3.191 and then when 100 is added to this result, the computer will end up with 103.2. On the other hand, if the computer adds 3.141 and 100 together first and gets the result of 103.1, when it tries to add the 0.05 to this number, the result will still be 109.1. The addition of 0.05 had no effect at all.

For the evaluation of *f()*, the lack of associativity of addition and the inverse square nature of each term of *f()* interact very badly. Say there is a binary star system near the edge of a galaxy (a common occurrence). The force between the two stars in the system will be very large, but all the stars in the center of the galaxy or on the other side of the galaxy will each have a very small effect. When the computer is adding together the terms in the force function, if it adds together the force from the other binary star first, all the other stars in the galaxy may, individually, not be significant enough to change the final force. The result would be that the binary star system would act as if it was not in the galaxy at all and just drift off. On the other hand, if the other binary star is added in last, the result will be much more accurate. (10:5-6)

A second type of problem with rounding can occur when two numbers of opposite signs are added together. If there are two numbers which are exactly represented by 8.1414926 and -8.1413867, the computer will have to represent them as 8.141 and -8.141. If you add these two numbers together, the exact result should be 0.001059 which can be exactly represented with 4 digits, but the computer will calculate the result as being zero.

As a general rule, if the computer has to add up a long list of numbers, the programmer should make sure that the numbers with the smallest absolute value are added together first. There shouldn't be a running total kept, instead when two numbers are added together, the result should just be put back into the list and the two numbers should be deleted. The computer should then try to add the next two smallest numbers together, and the process repeats until there is just one number left in the list.

The result is that even the "simple" particle-particle evaluation method must be made more complicated than it might be first expected to be. This additional sorting of the terms in the evaluation of the force function would make the Particle-Particle method an $O(n^2 \log n)$ method. Most of the more sophisticated evaluation methods do something akin to a sort on the stars and end up adding the forces in such a way that these rounding errors are reduced, although they do that more for reasons of efficiency than for accuracy.

4.2 Distant Stars Can Be Grouped Together

Consider the case shown in FIGURE 24., instead of calculating the force to each star in a distant cluster of stars, it is much more efficient to calculate the center of mass of the cluster and apply the force function to the center of mass. Since the calculation of the center of mass only requires summing the positions and masses, and doesn't require the expensive division and square root evaluation, this can save a lot of CPU time. It can save even more if this same center of mass information can be used for other stars that are also a long ways away from the cluster.

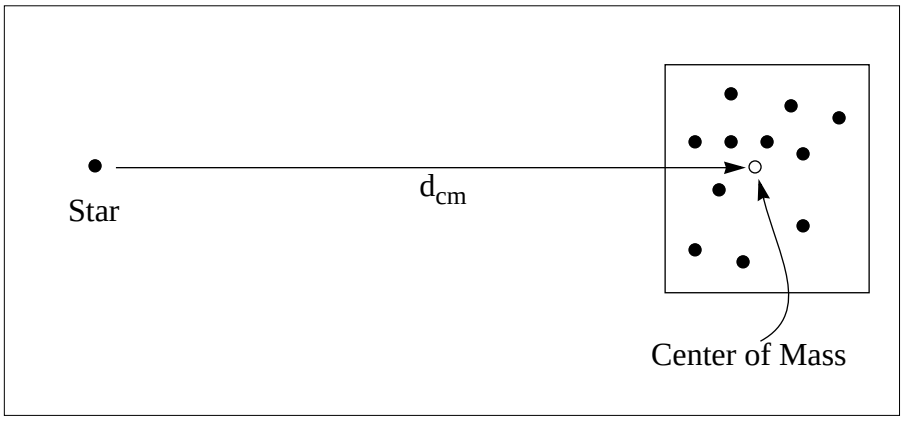


FIGURE 24. A Star and a distant cluster of stars

Replacing the cluster of stars with just the center of mass is only an approximation though, a certain amount of error will be created by doing this. The more distant the cluster and the more tightly packed the cluster is, the less the error will be. Deciding what should be considered "distant" and how tightly packet a set of stars is before it is considered a cluster are very important details that must be considered by any method that uses this technique. The error created by cluster the stars is somewhat offset by the reduced rounding error, so it is possible that this technique can actually increase the accuracy of *f()*.

Sources: (18:4-6)

4.3 Closely Packed Stars Can Be Moved at the Same Time

The opposite of the situation discussed in Section 4.2 is also an important special case. In FIGURE 25., we have a binary star system and several distant stars. For a star in the binary star system, the force function will be dominated by the other star and the effect of the other stars on the binary star's movement will be negligible. These two stars will just move in a Kepler (elliptical) orbit, but they will require a very small step size because they are orbiting so quickly (compare to other movements). Instead of calculate the force function for these two stars individually, we can replace these two stars with their center of mass and move the center of mass along. When these two stars get close enough to a third star to start making a difference, the point along the Kepler orbit can be calculated and the two stars can be broken apart again.

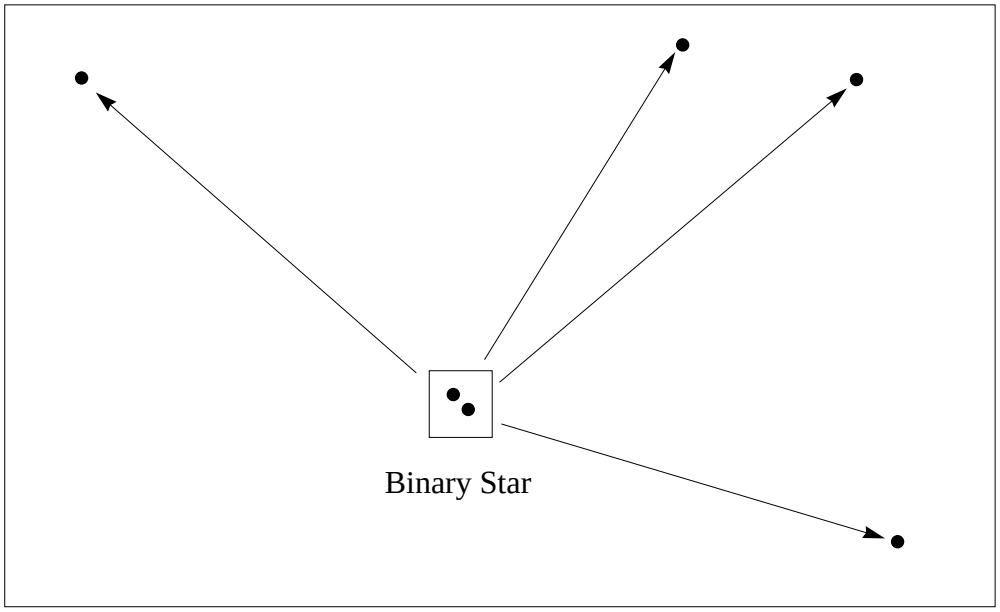


FIGURE 25. Binary star system and several distant stars

It isn't just binary stars that we can make a special case of. Even if two stars are moving too quickly to orbit one another and are just making a close pass, we can calculate the path along the conic section over a relatively large time step. No method of evaluating the force function can be faster than these cases where the force function isn't even evaluated.

Even 3 or 4 tightly packed stars can be considered as a special case. Again, the group of stars can be replaced by their center of mass for the purposes of moving the group, and the individual paths inside the group can be calculated by the simple Particle-Particle method. While this is an $O(n^2)$ operation, because n is so small, this is actually more efficient than other methods and keeps the errors caused by making simplifying assumptions from becoming too large.

Sources: (10:6,8,18:6)

4.4 Replace the Force Functions With a Potential Mesh

Instead of calculating the force that each star places on each other star, it is possible to calculate the force field that each star generates and apply this field to the universe. This is generally done by having all stars apply their masses to the mesh points in order to create a mass density function. The potential of the each individual star can be calculated by interpolating along the mesh points. By taking the gradient of this potential we can get the direction that is “down hill” from the star and also find how steep the “hill” is. This result is equivalent to the results of calculating the force function.

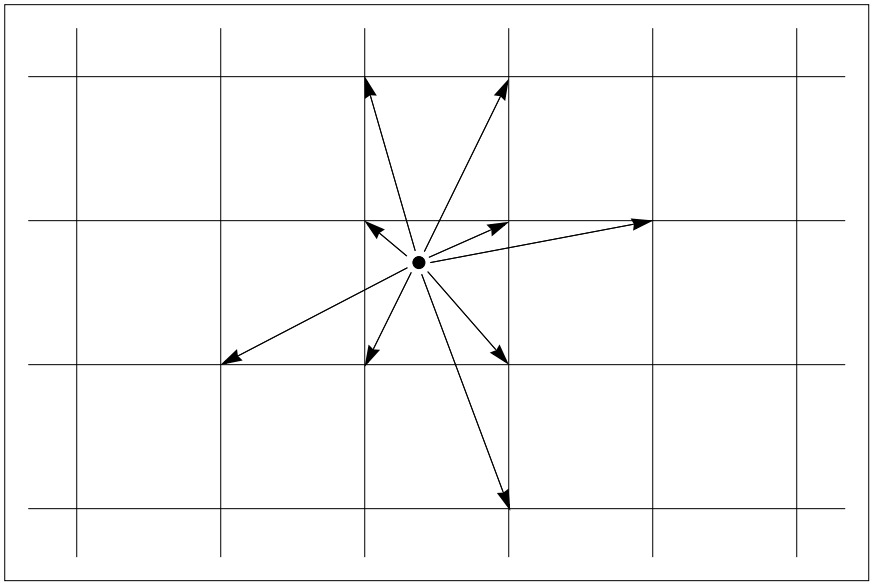


FIGURE 26. Applying a star’s potential to mesh points

This method replaces the calculation among every pair of particles with calculating the potential at many mesh points. This changes the evaluation method from being an $O(n^2)$ method to an $O(n \cdot g)$ method, where g is the number of mesh points. Due to the other calculations involved with this method, using a mesh is only useful if the number of mesh points that you must apply the star to is substantially less than the number of stars in the system. Since gravity is a very long range force, we end up having to apply the star to each mesh point in the universe. Other forces, such as electric or magnetic dipoles or short ranged forces such as the nuclear forces have to consider only the mesh points that are within a certain range of the body.

The accuracy of this method is very dependant on the number of mesh points, but the more mesh points used means more memory must be used. Since this method also spreads the body’s force over an area, this method does not give very accurate results when two objects are close together. Finally, the math involved in this method tends to get quite deep. It is hard to find formulas that can store the potential information *and* also have the gradient taken *and* also be accurate *and* also be quick to evaluate.

For the case of the gravitational N-body problem, it is possible to not have to apply the star to each mesh point at every step. Instead, only the nearest mesh points need to be updated every time, the ones that are further away can be updated less often. This is somewhat analogous to the discussion in Section 4.3 on page 47.

Sources: (8:3-4,6:1-10)

4.5 Creating Tree Structures For Evaluating the Force Function

The techniques described in Sections 4.2 and 4.3 can be applied in a hierarchial "tree" fashion. There are two basic approaches, namely bottom up and top down trees. Using either method, it is possible to create an force function evaluation method that is $O(n \log n)$ instead of $O(n^2)$, a significant improvement. The nodes in the trees are also prime locations to store information needed to implement variable step size ODE integration methods that have different step sizes for different stars. There are actually several variations of each of these types of trees, depending on how exactly the splitting and joining is carried out. The following are just typical examples.

Bottom up trees can be built by taking individual stars that are close together and replacing them with their centers of mass, and then taking the centers of mass and grouping them together. This process is repeated until only the center of mass of the entire system is left. The force function can be evaluated for individual stars by traversing the tree and if a particular branch of the tree is "distant", we can just use that particular nodes center of mass.

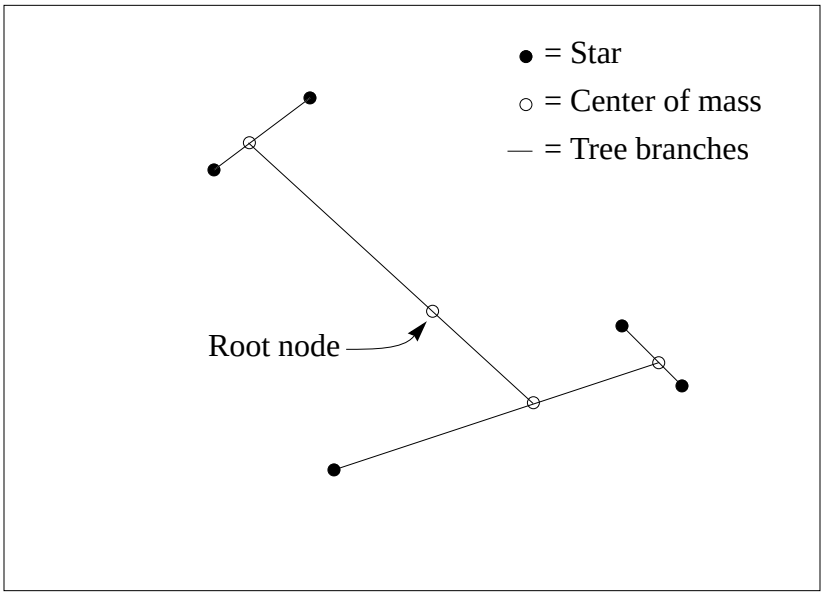


FIGURE 27. Bottom up tree for evaluating the force function

Top down trees are created by taking space and splitting it into regions (4 for the 2-D case, 8 for 3-D). If a region has more than one star, it is further sub-divided until each region contains only one star. Then, working back up the tree, the centers of mass for each region is calculated. The different regions of space are typically numbered with a special method called “oct codes” so that it is easier to find which regions are close to each other.

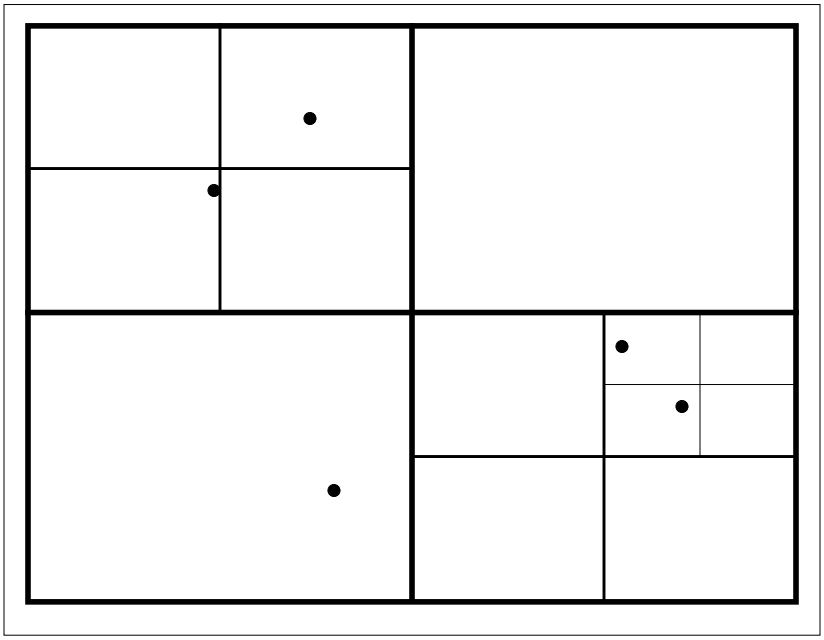


FIGURE 28. Top down tree for evaluating the force function

Sources: (18:13-15,8:5-7,5:4-8,6:11-16)

4.6 Hybrid Force Evaluation Methods

Due to the principle of superposition, it is possible to break the force into several parts, such as:

$$f(x(t)) = \text{near\_force} + \text{far\_force} + \text{external\_force}$$

It is then possible to use a different method to calculate each component of the force. For example, a mesh method could be used to create the far force and if you make sure the mesh is large enough so that there are substantially fewer mesh points than bodies, then you will have an *O(n)* routine to calculate the bulk of the stars. For medium range bodies, one of the tree methods could be used, giving an $O(n \log n)$ method for the bulk of the remaining bodies. When there are small clusters of only 3-6 bodies, the simple Particle-Particle method could be used. Finally, for close passes and binary systems, Kepler orbits can be used.

By using several of these methods, it is possible to create a system that is substantially faster and more accurate than any one method could possibly be. While I know of no implementation that uses all of these methods, most of the current research methods seem

to use at least two. The trick is to find combinations that work well together, where work on one method can be reused for another, or the effects of stars can be easily broken out so that they aren’t counted twice.

5.0 Analysis of the ODE Integration Methods

According to many books on numerical analysis, the multi-step Adam-Bashford method shouldn't even be in the running for the best method, and my taylor3 method should be a ignored. After all, the Adam-Bashford method isn't even as sophisticated as a predictor-corrector method and the Taylor series is rarely talked about except for its use as a fundamental theory.

A typical example of these opinions can be found in the highly regarded *Numerical Recipes in C* which has these comments on the subject:

> Runge-Kutta succeeds virtually always; but it is not usually fastest. Predictor-corrector methods, since they use past information, are somewhat more difficult to start up, but, for many smooth problems, they are computationally more efficient than Runge-Kutta. In recent years Bulirsch-Stoer has been replacing predictor-corrector in many applications, ... it appears that only rather sophisticated predictor-corrector routines are competitive.(14:568)

[The straight Adam-Bashford method can hardly be considered a "sophisticated" method...]

> The techniques described in this section [Bulirsch-Stoer] are not for differential equations containing non-smooth functions. ... A second warning is that the techniques in this section are not particularly good for differential equations which have singular points *inside* the interval of integration.(14:582)
>
> ...
>
> We suspect that predictor-corrector integrators have had their day, and that they are no longer the method of choice for most problems in ODEs. For high-precision applications, or applications where evaluations of the right hand sides are expensive, Bulirsch-Stoer dominates. For convenience, or for low-precision, adaptive-step size Runge-Kutta dominates. Predictor-corrector methods have been, we think, squeezed out in the middle. There is possibly only one exceptional case: high-precision solution of very smooth equations with very complicated right-hand sides, as we will describe later.(14:589)
>
> ...

> Our prediction is that, as extrapolation methods like Bulirsch-Stoer continue to gain sophistication, they will eventually beat out PC methods in all applications. We are willing, however, to be corrected.(14:592)

Let's look at the properties of our particular problem. First, the evaluation of the right-hand side, i.e. *f()*, is *very* expensive, even if a fast force evaluation method is used. The star system movement is normally very smooth, but it can also have singularities or poles in the complex plane when stars pass close to each other. The ODE integration routines need to be accurate, but since the results are just used to draw pictures, very high accuracy is not really required. All that is required is that it has to be accurate enough to avoid the worst of the movement errors.

Another atypical property of our star movement problem is that the smaller the step size is, the less a 'near collision' looks like a singularity and the more it looks like a smooth path. This is because the stars tend to move around each other so the point of the singularity (i.e., the other star) tends to move out of the way. Thus, by using a simpler method such as the Adam-Bashford method which requires a very smooth function, you can make the step size smaller and get a smooth function. If a more expensive method, such as the Gragg-Bulirsch-Stoer's is used, a much large step size must be used and thus you get the singularities that it can't handle.

The following two graphs, FIGURE 29. and FIGURE 30., show the x-axis component of the acceleration vector as a star makes a close pass to a collapsar. The time *t=0* is arbitrarily chosen as the time when the star is the closest to the collapsar.

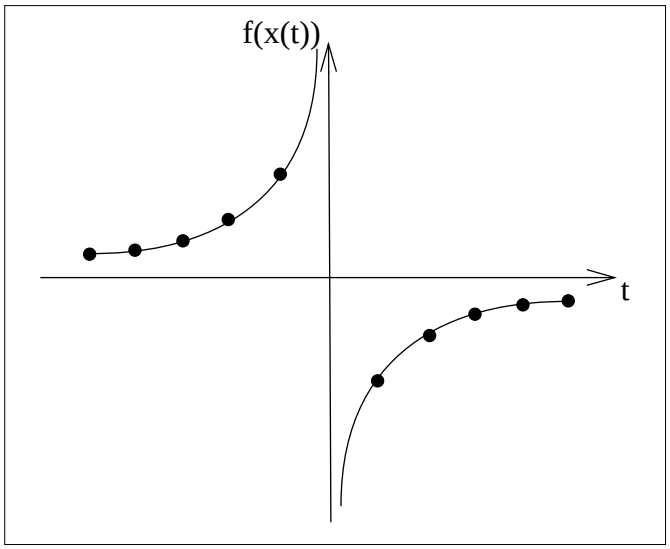


FIGURE 29. Graph of f() with a large step size.

Note the apparent singularity at t=0.

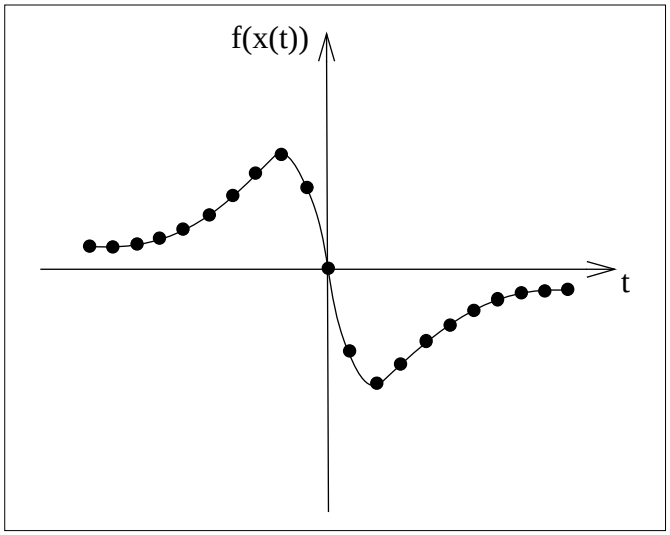


FIGURE 30. Graph of f() with a small step size.

Note that f() now looks like a continuous function and has smaller peaks.

The predictor-corrector methods seemed to be slightly worse than the Adam-Bashford method. The fact that *f()* has to be evaluated twice per step, and thus the predictor-corrector methods need twice the step size, seems to cancels out the improved error term. Apparently it is more important to have a smaller step size and thus make the function appear smoother than it is to have a more accurate, but larger step.

The Runge-Kutta method is hampered because it has to take trial steps into the future, but the movement of the stars changes how the gravitational force field will look in the future. So, the rk4 method has to not only calculate *f()* four times, it also has to do four trial star movements, which are not going to be completely accurate and is more expensive for the N-body problem than many other ODEs.

The taylor3 method that I developed is good at low accuracy levels because its error term has a very small constant. It also, by it's very structure, tends to cancel out the effects of the overstep phenomenon which is one of the worst forms of errors. It not just a coincidence that the taylor3 method is right at the borderline of being the most accurate method. It was the method that I used to develop most of XStar with and so, if I had noticed an excess amount of accuracy, I would have either increased the display rate or increased the number of stars. So, all the other methods are having to compete with the taylor3 method at the very peak of its efficiency.

6.0 Conclusion

XStar has certain characteristics that influence which methods should be used to solve the N-body problem. They include:

- The long “history” of locations that XStar generates
- The real-time display requirements of XStar
- The requirement of having high enough accuracy so that the worst of the deviations do not occur, but there is no requirement that the accuracy must be extremely high.
- Smooth paths are the norm
- For practical reasons, only a few to a few dozen stars can be display on the screen without it becoming cluttered.

These characteristics lead to corresponding conclusions:

- Multi-step methods are very practical
- Variable step size methods are much harder to implement
- In practice, it appears that a method with an order of at least 4 is needed
- Simple Adam-Bashford methods work well
- Complex force function evaluation methods are not required

These conclusions combine to form the final conclusion that the 7^{th} order Adam-Bashford method is the best method for most of the practical accuracy range. For very low accuracy, the taylor3 method is the best, and for the very high accuracy, the gpemce8 is the best.

References

1. Atkinson, Kendall E. 1978. An Introduction to Numerical Analysis. John Wiley & Sons, New York.

2. Burden and Faires. 1993. Numerical Analysis. 5th ed. PWS Publishing Company, Boston.

3. Cohen, I. Bernard. 1985. The Birth of a New Physics. W. W. Norton & company, Inc.

4. Conte, S. D., and Carl de Boor. 1980. Elementary Numerical Analysis. McGraw-Hill, New York

5. Demmel, Jim. 1995. CS267 Lecture 26 Notes. Internet URL: http://www.icsi.berkeley.edu/cs267/lecture26/lecture26.html

6. Demmel, Jim. 1995. CS267 Lecture 27 Notes. Internet URL: http://www.icsi.berkeley.edu/cs267/lecture27/lecture27.html

7. Feynman, Richard. 1965. The Character of Physical Law. The M.I.T. Press, Cambridge, MA.

8. Graps, Amara. 1995. Amara's Recap of Particle Simulation Methods. Internet URL: ftp://ftp.amara.com/papers/nbody.txt.

9. Hennesy, John L. and David A. Pattterson. 1990. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers, Inc. San Mateo California.

10. Hut, Peit, Steve McMillan, Junichiro Makino. 1993. Starlab Primer. Internet URL: http://www.sns.ias.edu/~starlab/Primer.ps

11. Kersten, Dr. Leedert. 1996. Numerical Methods, 2nd ed. Lecture Notes for EM480. University of Nebraska Press. Lincoln Nebraska.

12. Marciniak, Andrzej. 1985. Numerical Solutions of the N-body Problem. D. Reidel Publishing Company. Dordrecht/Boston/Lancaster.

13. Pizer, Stephen M. 1983. To Compute Numerically. Little, Brown & Company, Boston.

14. Press, William H., et al. 1989 Numerical Recipes in C. Press Syndicate of the University of Cambridge, New York.

15. Scheid, Francis. 1968. Theory and Problems of Numerical Analysis. McGraw-Hill Book Company, New York.

16. Swokowski, Earl William. 1991. Calculus. 5th ed. PWS-Kent Publishing Company. Boston.

17. Tipler, Paul A. 1991. Physics for Scientists and Engineers. Worth Publishers, New York.

18. Warren, Michael S. and John K Salmon. 1994. A Portable Parallel Particle Program. Internet URL: http://qso.lanl.gov/papers/cpc/v9.ps

19. Zill, Dennis G. A First Course in Differential Equations with Applications. PWS Publishers, Boston.